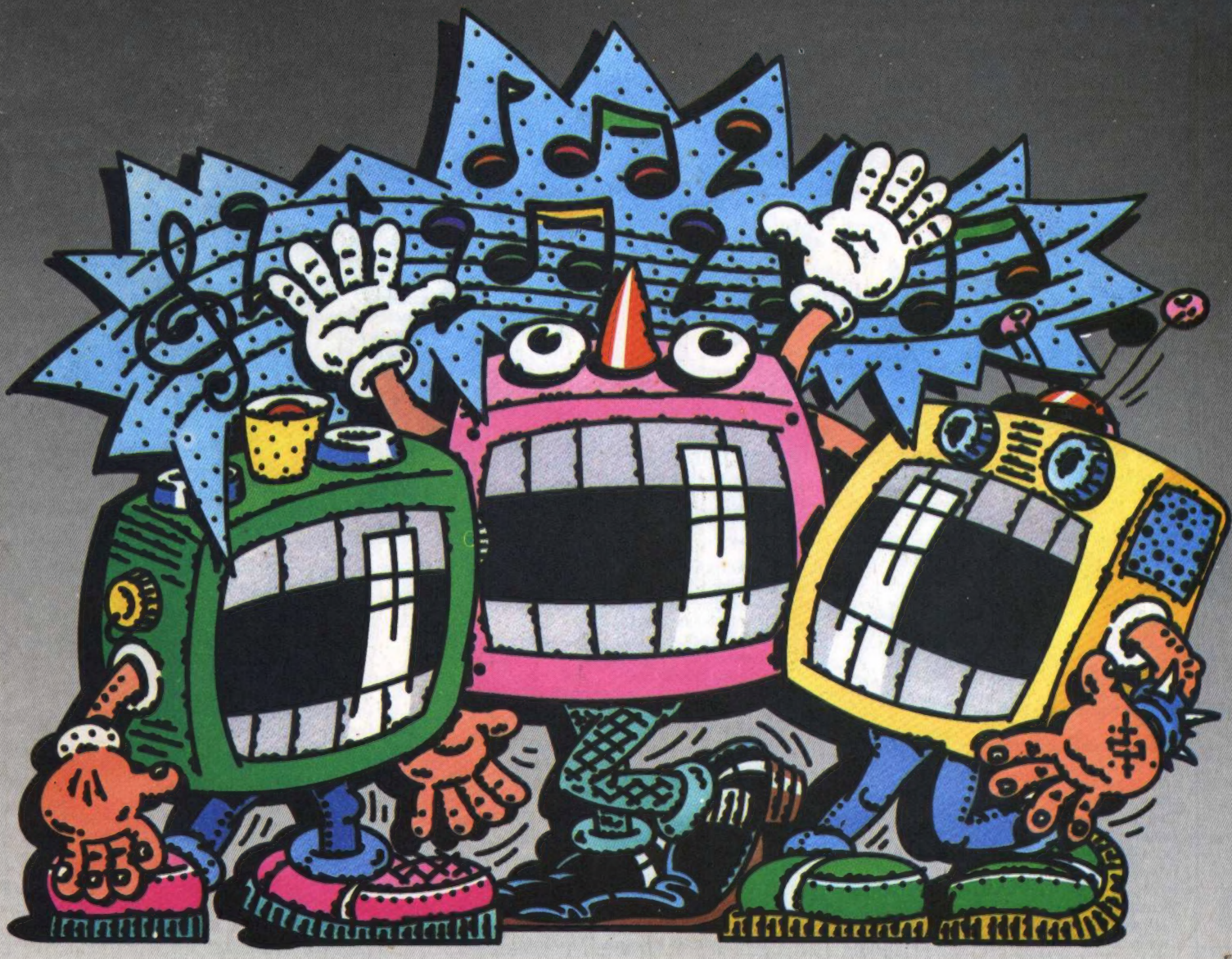
CURSO PRÁTICO 38 DE PROGRAMAÇÃO DE COMPUTADORES

INPUT

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

NOVA CULTURAL

Cz$ 20,00



INPUT

Vol. 3 N.º 38

NESTE NÚMERO

PROGRAMAÇÃO BASIC

PROGRAMAÇÃO DE MELODIAS NO MICRO

Notação musical. Tonalidade. Cálculo do valor da duração e da tonalidade de uma nota. Um gabarito musical. Bemóis, sustenidos, acidentes fixos. Ritmo. Tradução de uma melodia. Andamento. Tabela de conversão..... 741

CÓDIGO DE MÁQUINA

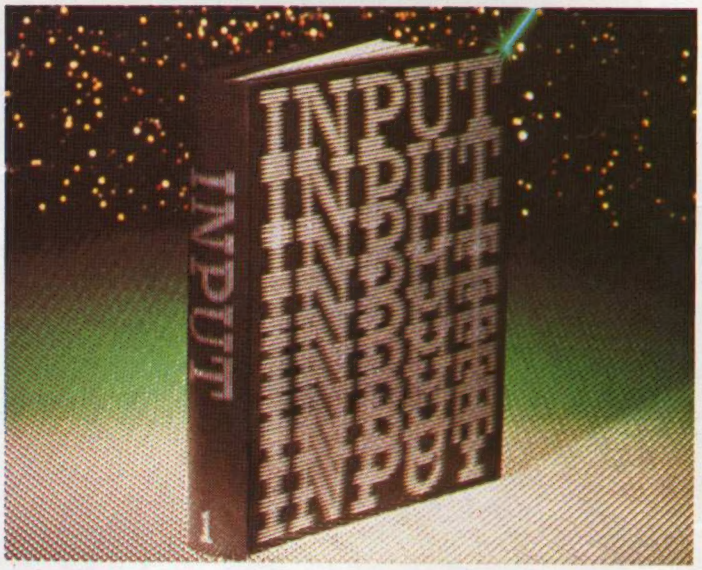
UM VIDEOGAME EM ASSEMBLER

Estrutura geral do jogo. A estrutura do programa. Programas Basic que auxiliam a montagem. Os códigos. Tabelas de dados em ASCII. Criação das quatro telas. Rotinas da ROM. Impressão de caracteres na tela. Programação de pausas. Limpeza da tela. Correção de erros 748

PROGRAMAÇÃO DE JOGOS

O JOGO DO OTELO (1)

Estrutura e regras do jogo. Dicas para vencer. A primeira tela. Seleção dos gráficos para o tabuleiro e as peças. Inicialização do jogo. Digitando a rotina principal. A vez do jogador. Rotina para desenhar letras..... 756



PLANO DA OBRA

"INPUT" é uma obra editada em fascículos semanais, e cada conjunto de 15 fascículos compõe um volume. A capa para encadernação de cada volume estará à venda oportunamente.

COMPLETE SUA COLEÇÃO

Exemplares atrasados, até seis meses após o encerramento da coleção, poderão ser comprados, a preços atualizados, da seguinte forma: 1. PESSOALMENTE — Por meio de seu jornaleiro ou dirigindo-se ao distribuidor local, cujo endereço poderá ser facilmente conseguido junto a qualquer jornaleiro de sua cidade. Em **São Paulo**, os endereços são: rua Brigadeiro Tobias, 773, Centro; av. Industrial, 117, Santo André; e no **Rio de Janeiro**: rua da Passagem, 93, Botafogo. 2. POR CARTA — Poderão ser solicitados exemplares atrasados também por carta, que deve ser enviada para DINAP — Distribuidora Nacional de Publicações — Números Atrasados — Estrada Velha de Osasco, 132, Jardim Teresa — CEP 06000 — Osasco — SP. Não envie pagamento antecipado. O atendimento será feito pelo reembolso postal e o pagamento, incluindo as despesas postais, deverá ser efetuado ao se retirar a encomenda na agência do Correio. 3. POR TELEX — Utilize o n.º (011) 33 670 DNAP.

Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações, Lda. — Qta. Pau Varais, Azinhaga de Fetais — 2 685, Camarate — Lisboa; Apartado 57 — Telex 43 069 JARLIS P.

Atenção: Após seis meses do encerramento da coleção, os pedidos serão atendidos dependendo da disponibilidade do estoque.

Obs.: Quando pedir livros, mencione sempre título e/ou autor da obra, além do número da edição.

COLABORE CONOSCO

Encaminhe seus comentários, críticas, sugestões ou reclamações ao Serviço de Atendimento ao Leitor — Caixa Postal 9442, São Paulo — SP.

Editor
VICTOR CIVITA

REDAÇÃO
Diretora Editorial: Iara Rodrigues

Editor Executivo: Antonio José Filho

Editor Chefe: Paulo de Almeida
Editor de Texto: Cláudio A. V. Cavalcanti
Chefe de Arte: Carlos Luiz Batista
Assistentes de Arte: Ailton Oliveira Lopes, Dilvacy M. Santos, Grace Alonso Arruda, José Maria de Oliveira, Monica Lenardon Corradi
Secretária de Redação/Coordenadora: Stefania Crema
Secretários de Redação: Beatriz Hagström, José Benedito de Oliveira Damião, Maria de Lourdes Carvalho, Marisa Soares de Andrade, Mauro de Queiroz

COLABORADORES
Consultor Editorial Responsável: Dr. Renato M. E. Sabbatini (Diretor do Núcleo de Informática Biomédica da Universidade Estadual de Campinas)
Execução Editorial: DATAQUEST Assessoria em Informática Ltda., Campinas, SP
Tradução: Reinaldo Cúrcio
Tradução, adaptação, programação e redação: Abílio Pedro Neto, Aluísio J. Dornellas de Barros, Marcelo R. Pires Therezo, Raul Neder Porrelli
Coordenação Geral: Rejane Felizatti Sabbatini
Editora de Texto: Ana Lúcia B. de Lucena
Assistente de Arte: Dagmar Bastos Sampaio

COMERCIAL
Diretor Comercial: Roberto Martins Silveira
Gerente Comercial: Flávio Maculan
Gerente de Circulação: Denise Maria Mozol

PRODUÇÃO
Gerente de Produção: João Stungis
Coordenador de Impressão: Atílio Roberto Bonon

Preparador de Texto/Coordenador: Eliel Silveira Cunha
Preparadores de Texto: Alzira Moreira Braz, Ana Maria Dilguerian, Karina Ap. V. Grechi, Levon Yacubian, Luciano Tasca, Maria Teresa Galluzzi, Maria Teresa Martins Lopes, Paulo Felipe Mendrone
Revisor/Coordenador: José Maria de Assis
Revisoras: Conceição Aparecida Gabriel, Isabel Leite de Camargo, Ligia Aparecida Ricetto, Maria de Fátima Cardoso, Nair Lúcia de Brito
Paste-up: Anastase Potaris, Balduino F. Leite, Edson Donato

© Marshall Cavendish Limited, 1984/85.
© Editora Nova Cultural Ltda., São Paulo, Brasil, 1986.

Edição organizada pela Editora Nova Cultural Ltda.
Av. Brigadeiro Faria Lima, n.º 2000 - 3.º andar
CEP 01452 - São Paulo - SP - Brasil
(Artigo 15 da Lei 5 988, de 14/12/1973).
Esta obra foi composta na AM Produções Gráficas Ltda. e impressa na Divisão Gráfica da Editora Abril S.A.

PROGRAMAÇÃO DE MELODIAS NO MICRO

- CONVERSÃO DE PARTITURAS EM PROGRAMAS
- ACIDENTES
- RITMO
- EXECUÇÃO DE MELODIAS

Seu micro é capaz de executar, sozinho, melodias inteiras. Aprenda a converter partituras em linhas de programas BASIC e ouça, então, suas canções prediletas.

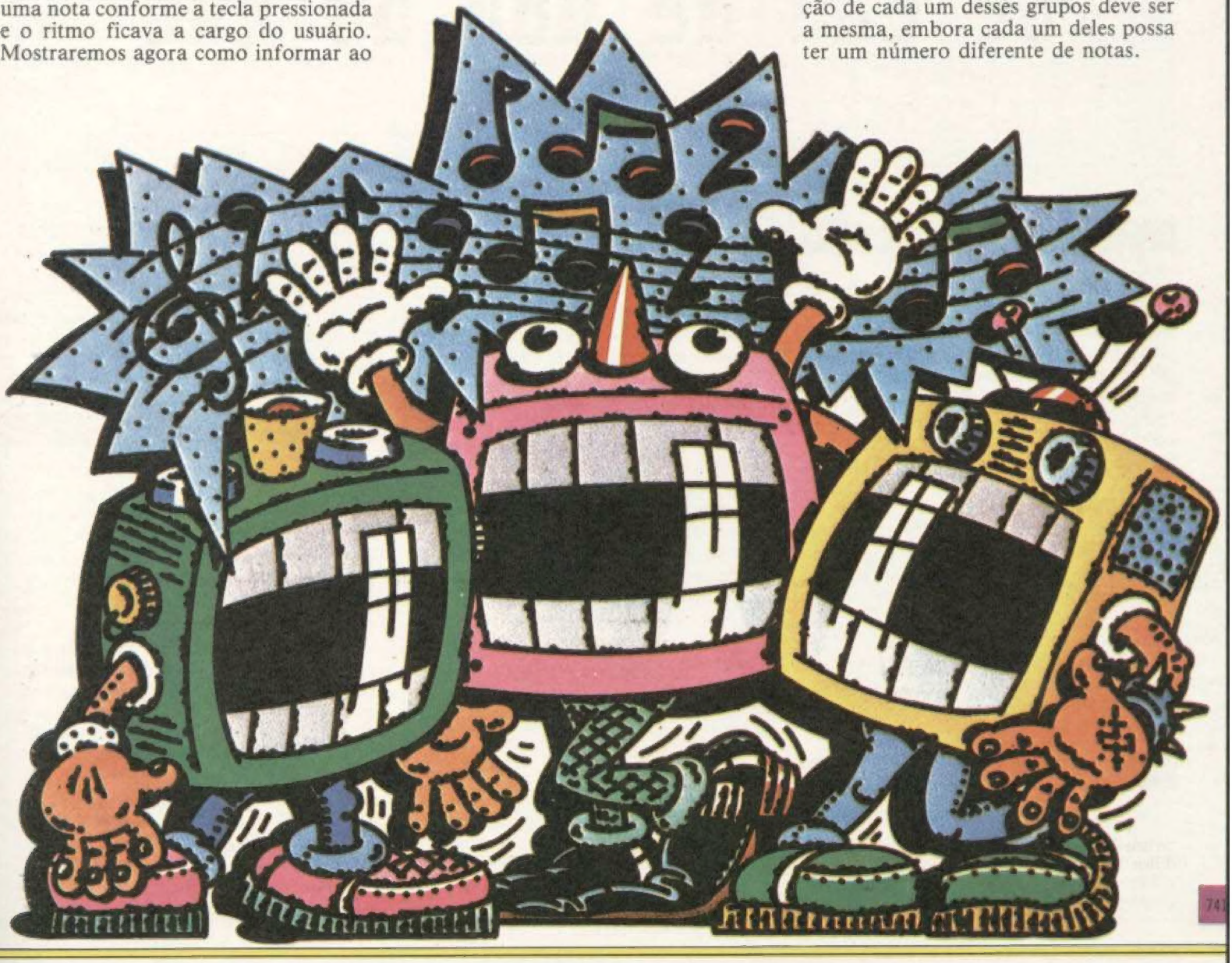
A música compõe-se de dois elementos básicos: som e ritmo. Nosso primeiro artigo sobre música em micros (veja página 721) limitou-se à produção de notas musicais. O computador emitia uma nota conforme a tecla pressionada e o ritmo ficava a cargo do usuário. Mostraremos agora como informar ao computador o ritmo desejado. Além disso, daremos as noções de notação musical necessárias à tradução de qualquer partitura, o que lhe permitirá executar no micro sua melodia predileta.

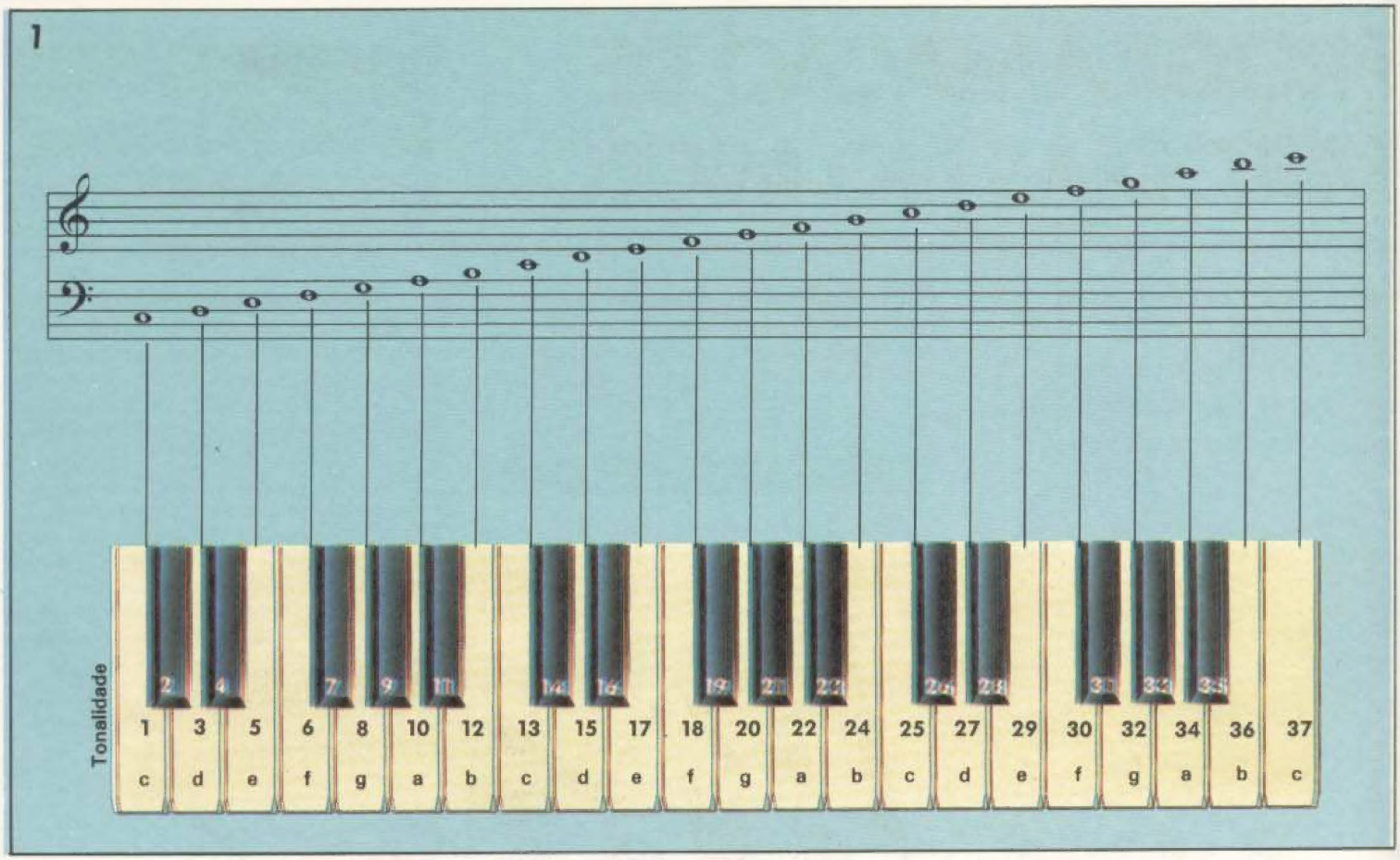
NOTAÇÃO MUSICAL

Em geral, as partituras musicais são escritas em um ou dois grupos de cinco linhas horizontais, paralelas e eqüidistantes, denominados *pauta*. Sobre uma ou entre duas linhas colocam-se os símbolos que representam as notas. A posição destas na pauta, no sentido vertical, determina a tonalidade; no sentido horizontal, a ordem de execução. O formato dos símbolos, por sua vez, determina a duração da nota.

Notas alinhadas verticalmente devem ser tocadas ao mesmo tempo, em acordes; estes, porém, não foram incluídos em nossos programas, que só tocam uma nota de cada vez.

Linhas verticais, as *barras* dividem a música em grupos de notas. A duração de cada um desses grupos deve ser a mesma, embora cada um deles possa ter um número diferente de notas.





TONALIDADE

O programa que apresentamos no artigo anterior transformou seu computador em uma espécie de instrumento musical. Como você já deve ter observado, ele funciona muito bem. Contudo, o número de melodias que pode executar é reduzido, em razão do pequeno número de notas disponíveis. Além disso, o usuário, que, muitas vezes, jamais tocou qualquer tipo de instrumento, é obrigado a executar cada música.

Pode-se facilmente converter uma partitura nos números que o computador precisa para tocá-la, mesmo sem ter grandes informações sobre música. Na página 743, fornecemos uma tabela de conversão justamente com esta finalidade. Antes, porém, devemos ampliar um pouco nossos conhecimentos a respeito de notação musical.

LER EM VEZ DE OUVIR

A figura 1 exibe um trecho de partitura. Nela estão representadas todas as notas disponíveis no teclado, em uma série crescente — é a chamada *escala cromática*.

Os símbolos no início das pautas são denominados *claves*: o da pauta de cima é a *clave de sol*, mais aguda; o da pauta de baixo, a *clave de fá*, mais grave. As claves determinam a tonalidade da pauta. Sem uma clave, as mesmas notas poderiam ser tocadas numa tonalidade mais alta ou mais baixa.

Os símbolos das notas ficam sempre entre duas ou exatamente sobre uma das linhas da pauta. Para escrever as notas que ficam acima ou abaixo destas, usam-se *linhas suplementares*. Na figura 1, por exemplo, a oitava e as três últimas notas estão desenhadas sobre linhas suplementares.

Para colocar as notas da partitura em um programa BASIC é necessário calcular os valores da duração bem como de tonalidade correspondentes a cada uma delas. Qualquer pessoa pode se confundir nessa demorada tarefa. Mas não desanime: há uma maneira mais fácil de realizá-la.

Voltemos à figura 1: observe que cada tecla foi marcada com um valor de 1 a 37. Esses números serão usados para calcular aqueles que, efetivamente, farão parte do programa.

Poderíamos consultar a mesma figura sempre que quiséssemos "traduzir" uma melodia, mas o processo de comparação entre a partitura e o desenho também seria bastante cansativo. Para simplificar o trabalho, o mais prático seria o uso de um dispositivo ou "gabarito" que, colocado sobre qualquer nota de partitura, mostrasse o valor correspondente à sua tonalidade.

UM GABARITO MUSICAL

Não há nenhum segredo na construção de um dispositivo desse tipo: precisamos simplesmente desenhar uma escala musical numa folha de papel, o que leva alguns poucos minutos. Provavelmente, melodias diferentes exigirão gabaritos diferentes, já que o tamanho das pautas pode variar.

A primeira coisa a fazer é pôr o papel sobre a partitura e copiar as linhas das pautas, indicando todas as possíveis posições verticais de uma nota. Em seguida, basta assinalar o valor correspondente a cada marca. Para identificar os valores que deveremos usar nos programas, colocamos o gabarito sobre a partitura e lemos nota por nota com sua ajuda.

Os valores correspondentes a cada tonalidade estão representados na tabela de conversão. O Spectrum, o Apple e o

TABELA DE CONVERSÃO

Escala principal	Spectrum	Apple	TK-2000	TRS-Color	MSX
1	−12	192	192	C	C
2	−11	182	182	C+	C+
3	−10	172	172	D	D
4	−9	162	162	D+	D+
5	−8	154	154	E	E
6	−7	146	146	F	F
7	−6	137	137	F+	F+
8	−5	128	128	G	G
9	−4	121	121	G+	G+
10	−3	114	114	A	A
11	−2	108	108	A+	A+
12	−1	102	102	B	B
13	0	96	96	C	C
14	1	90	90	C+	C+
15	2	85	85	D	D
16	3	80	80	D+	D+
17	4	76	76	E	E
18	5	72	72	F	F
19	6	67	67	F+	F+
20	7	64	64	G	G
21	8	60	60	G+	G+
22	9	56	56	A	A
23	10	53	53	A+	A+
24	11	50	50	B	B
25	12	47	47	C	C
26	13	45	45	C+	C+
27	14	42	42	D	D
28	15	40	40	D+	D+
29	16	37	37	E	E
30	17	35	35	F	F
31	18	33	33	F+	F+
32	19	31	31	G	G
33	20	29	29	G+	G+
34	21	28	28	A	A
35	22	26	26	A+	A+
36	23	25	25	B	B
37	24	23	23	C	C

TK-2000 usam números, enquanto o MSX e o TRS-Color usam letras. Para saber o que escrever junto a cada marca do gabarito, compare os números da figura 1 com os da tabela e procure o valor correspondente na coluna dedicada ao seu micro.

BEMÓIS E SUSTENIDOS

Nem todas as melodias são escritas na escala "C maior" ("dó maior", como vimos no artigo anterior); portanto, às vezes precisaremos traduzir os bemóis e sustenidos das partituras.

Se observarmos os acidentes fixos — explicados mais adiante —, distinguiremos os bemóis ou sustenidos da canção. Poderemos, assim, substituir o valor da nota natural no gabarito pelo valor do bemol ou do sustenido.

Bemóis e sustenidos são as teclas pretas, e o valor de suas tonalidades está na tabela. O menor valor de um C sustenido é 2 (usando os números da escala principal da tabela, você descobrirá o valor correspondente em sua máquina). D bemol tem o mesmo valor, que está entre 1 e 3 (valores de C e D, respectivamente).

Símbolos para bemóis e sustenidos não foram introduzidos na figura 1 para não torná-la confusa. Quando se quer representar o sustenido de uma nota, ela aparece precedida pelo símbolo correspondente (parecido com o caracter # "cardinal" de seu computador). Da

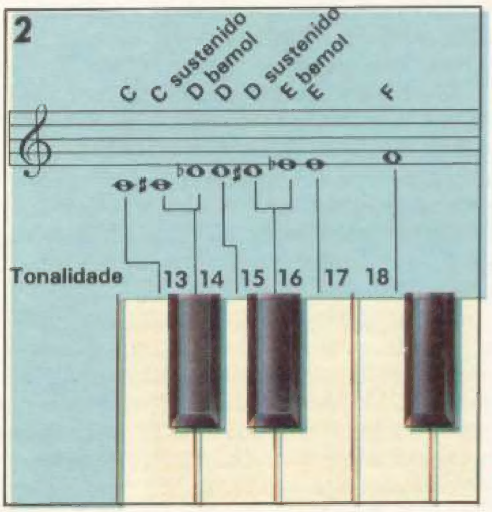
Como o computador produz sons?

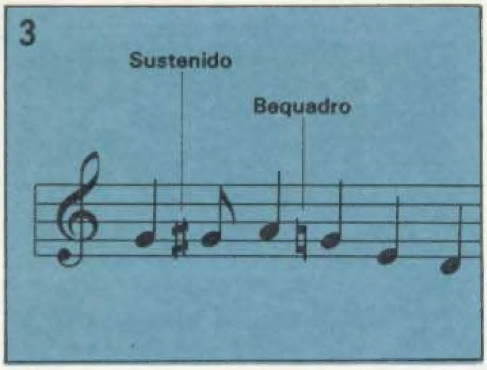
Para que um dispositivo digital, como o computador, possa gerar um fenômeno analógico (continuamente variável em intensidade), como o som, é necessário ligá-lo a um aparelho especial, chamado *conversor digital-analógico* (DA). Este tem a função de converter uma lista de números armazenada ou gerada no computador em uma onda continuamente variável.

No nosso caso, cada número da lista representa a *amplitude* ou intensidade da onda em um determinado momento. A lista de números, com valores de amplitude tomados a intervalos regulares (geralmente milissegundos), constitui, assim, uma amostragem da onda a ser produzida. O inverso do intervalo de amostragem, denominado freqüência de amostragem, dependerá da freqüência máxima que o som deve atingir.

O conversor analógico-digital (AD) funciona de modo inverso ao conversor DA, transformando uma onda analógica em uma lista de números. Esse dispositivo é utilizado, por exemplo, em sistemas de reconhecimento de voz para microcomputadores ou na conversão de entradas provenientes de joysticks analógicos.

O conversor DA só pode ser empregado nos micros que dispõem de geradores internos de sons, como o TRS-Color, o MSX e o Spectrum. Colocando-se vários conversores DA lado a lado, a placa geradora de som desses micros torna-se capaz de emitir várias notas ao mesmo tempo.





mesma maneira, símbolos de bemol (parecidos com um b minúsculo) precedem as notas correspondentes.

A figura 2 mostra as primeiras notas "C maior", incluindo bemóis e sustenidos, na clave de sol. Observe que C sustenido e D bemol são a mesma nota, mas ocupam posições verticais diferentes na pauta. Notas isoladas podem, assim, representar bemóis e sustenidos, desde que sejam precedidas pelos símbolos correspondentes — denominados *acidentes*. O efeito de um acidente dura, para a nota que o segue, até a próxima barra, a menos que seja cancelado antes do fim do compasso. É um terceiro símbolo, chamado *bequadro*, que cancela o acidente e restaura a tonalidade natural da nota. A figura 3 mostra um trecho de partitura onde aparece um G sustenido. O G seguinte tem sua tonalidade natural preservada pelo bequadro que o precede.

ACIDENTES FIXOS

Quando uma canção é executada em uma escala que inclui um bemol ou um sustenido, essa nota será usada o tempo todo, no lugar do valor natural correspondente. Para evitar a repetição do símbolo, ele é colocado após a clave, no início da pauta. Nesse local, o símbolo é chamado de *acidente fixo* e, por meio dele, pode-se identificar a escala em que é tocada a canção.

A ausência de um acidente fixo indica que a escala é "C maior", e, ao contrário, qualquer acidente fixo indica que se trata de outra escala. Se, por exemplo, uma pauta apresenta um sustenido como acidente fixo na linha do F, todos os F, em todas as oitavas, serão sustenidos.

No artigo anterior, mostramos que um F sustenido é necessário para preservar o arranjo de intervalos na escala "B maior". Um sustenido na posição do F como acidente fixo indica ao músico que a melodia foi escrita em "B maior". Não se esqueça, portanto, de modificar no gabarito os valores das tonalidades correspondentes, sempre que a partitura tiver acidentes fixos.

A figura 4 mostra um trecho de partitura em "F maior". Nele vemos um acidente fixo bemol. Os valores das tonalidades estão indicados: note que o da quarta nota, um B bemol, é 23. Aqui o gabarito revela toda a sua utilidade. Se a música que está sendo transcrita tem um B bemol como acidente fixo, colocamos o valor correspondente ao bemol na marca do B. Assim, a tradução dos acidentes se torna tão fácil quanto a das notas naturais.

Mas tenha cuidado: muitas vezes alguns acidentes isolados — e naturais isolados — têm prioridade sobre os acidentes fixos.

RITMO

Depois de calcular a tonalidade das notas, devemos cuidar da duração de cada uma delas e da tradução das pausas, que têm seus próprios símbolos. A figura 5 mostra os símbolos das notas e das pausas com duração equivalente e o valor relativo da duração.

Durações relativas são utilizadas para que um *andamento* — velocidade de execução — defina, posteriormente, a verdadeira duração. Podemos, assim, experimentar diversos andamentos, até que um deles nos soe melhor. O programa do final do artigo permitirá esse tipo de ajuste.

As pausas mais longas são representadas por meio de pequenos retângulos pretos, desenhados acima ou abaixo da linha média da pauta. Quando um ponto sucede uma nota ou uma pausa, sua duração é multiplicada por um e meio. Uma nota pontuada duas vezes tem uma duração uma vez e três quartos maior que a nota sem pontos.

Além do acidente fixo, junto à clave aparecem dois algarismos, dispostos como fração numérica. O de baixo indica a espécie de notas em que o compasso está dividido; o de cima, o número dessas notas no compasso. Na figura 6, por exemplo, vemos a fração 6/8 depois da clave, o que significa que há seis colcheias por compasso.

Para nossos fins essas noções de notação musical são suficientes: vamos apenas transcrever a partitura para criar um programa.

TRADUÇÃO DE UMA MELODIA

Vamos agora traduzir uma pequena partitura em números (ou letras, conforme sua máquina) que o computador possa utilizar para determinar tonalidades e durações de notas.

A figura 6 mostra a partitura de uma canção infantil inglesa, *Three Blind Mice* (*Três Ratinhos Cegos*) acompanhada dos números que representam a tonalidade na escala principal, previamente definida. Para identificar os números correspondentes em seu computador, use a tabela de conversão. Os valores para a duração relativa também estão indicados.

Observando a partitura em questão, você verá que, sempre que três colcheias (veja figura 5) aparecem juntas, elas são ligadas por um traço horizontal. Não estranhe: a única finalidade dessa forma de notação — não obrigatória — é facilitar a leitura. As notas permanecem exatamente as mesmas, não havendo modificações nem na altura nem na duração.

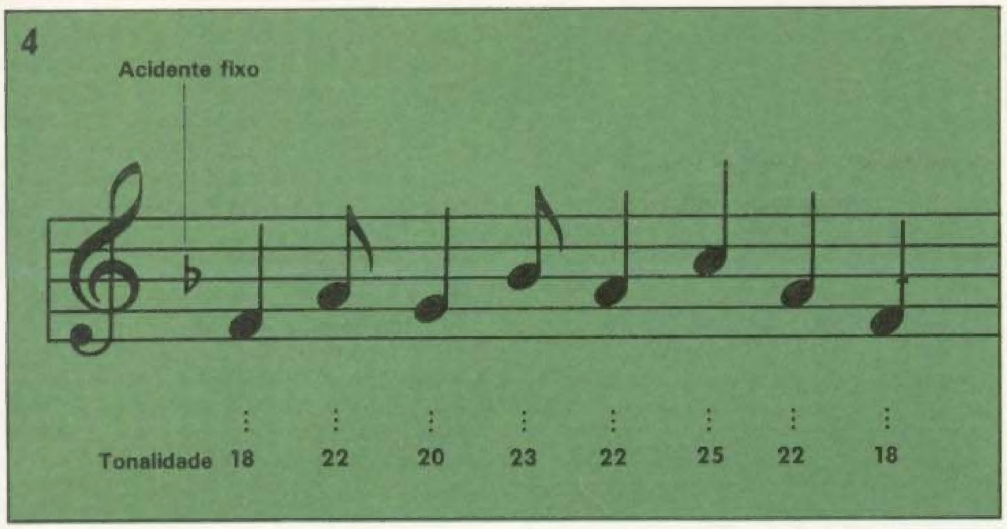
Já no oitavo compasso, há duas notas ligadas, mas por uma linha curva, e não reta. Nesse caso há mudança: a linha, chamada *ligadura*, indica que a segunda nota não deve soar separadamen-

MICRO DICAS

SELEÇÃO DE PARTITURAS

Se você sabe ler partituras, não terá a menor dificuldade em encontrar as músicas de sua preferência para "tocar" no computador, usando o programa deste artigo. Mas, se você dispõe apenas de noções elementares de notação musical, precisará contornar suas deficiências selecionando partituras simplificadas. Uma excelente fonte são os álbuns de exercícios musicais para iniciantes de instrumentos menos complexos, como a flauta doce. Neles, melodias não harmonizadas (isto é, sem acordes) são apresentadas na forma de partituras tão simples que você não terá dificuldade nenhuma em acompanhar.

A seleção de músicas será um pouco mais difícil para quem não lê absolutamente nada de uma partitura. Mas, ainda assim, resta o popular recurso de "tocar por números". Alguns fabricantes de instrumentos musicais para crianças publicam partituras em que as notas são acompanhadas por números. Estes indicam a tecla a ser pressionada. Estabelecendo a correspondência entre os números e as teclas do computador, você poderá executar músicas completas sem saber sequer o que é um "dó".



te, ou seja, que o som da primeira se prolongará pelo tempo correspondente à duração de ambas as notas. Poderíamos ter, também, ligaduras com várias notas. Para transcrevê-las em dados computáveis, bastaria somar as durações correspondentes, o que nos daria uma nota mais longa.

LINHAS DATA

Os números das tonalidades e durações devem ser colocados em linhas **DATA**, para que o programa possa obtê-los com **READ** no decorrer da execução. Os números correspondentes à canção da figura 6 foram incorporados ao programa dessa maneira. Cada linha contém dois compassos (exceto no TRS-Color e no MSX), o que torna mais fácil a compreensão das linhas **DATA**.

A duração total de cada compasso deve ser a mesma — doze unidades, no nosso caso. Assim, se você transcrever uma música e o ritmo parecer incorreto, verifique se cada compasso tem a mesma duração, corrigindo os erros encontrados. Use as funções de edição de seu computador para duplicar compassos iguais ou semelhantes, economizando tempo de digitação.

ANDAMENTO

Os valores da duração de cada nota são relativos, já que podemos variar o andamento (velocidade) que se imprime à execução de um trecho da melodia. O programa solicita um valor para o andamento, logo que é executado. Como valores menores correspondem a maiores velocidades de execução, podemos considerar o valor que fornecemos como o inverso do andamento.

Há um detalhe importante sobre o funcionamento do programa que se aplica a todos os micros. Suponhamos que a melodia contenha um compasso com uma única nota bem longa, seguido por outro compasso com várias notas curtas. No segundo caso, as linhas que lêem e testam os dados são executadas mais vezes que no primeiro. Devido ao tempo de operação extra, a velocidade de execução do segundo compasso será um pouco menor. Deve-se imprimir, portanto, a maior velocidade possível à execução dessas linhas BASIC. Por isso, não há no programa linhas que testem o fim dos dados: o programa termina simplesmente com uma mensagem de erro indicando que não há mais dados.

Se, porém, ao traduzir suas músicas prediletas, você quiser um final mais elegante, coloque um laço **FOR...NEXT** que leia exatamente a quantidade de números que as linhas **DATA** contêm.

Seguem-se os programas que executam a partitura da figura 6.

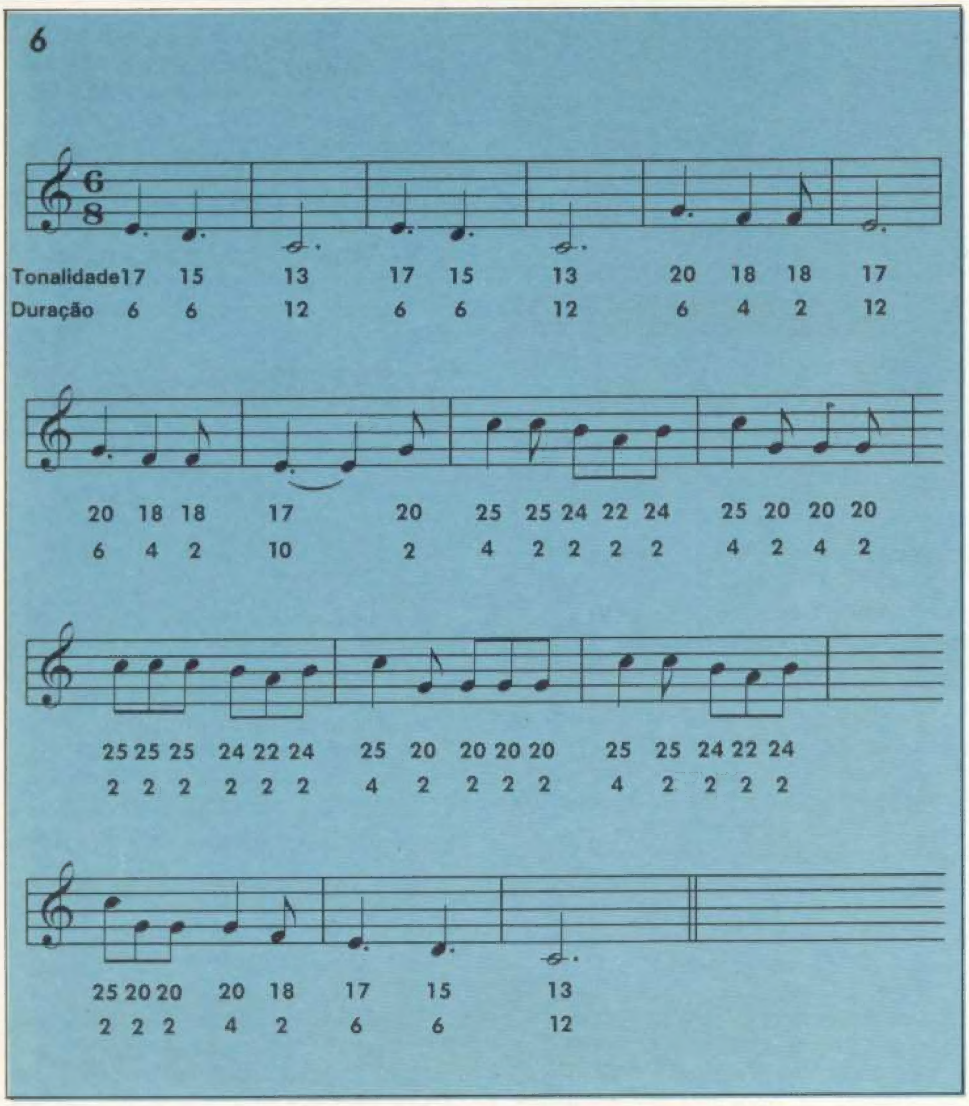
```
10 INPUT "TEMPO (1-50)",t:
LET t=t/100
20 IF t<0.001 OR t>0.5 THEN
GOTO 10
30 FOR n=0 TO 1 STEP 0: READ
a,b: SOUND a*t,b: NEXT n
100 DATA 6,4,6,2,12,0,6,4,6,2,
12,0
110 DATA 6,7,4,5,2,5,12,4,6,7,
4,5,2,5,10,4,2,7
120 DATA 4,12,2,12,2,11,2,9,2,
11
130 DATA 4,12,2,7,4,7,2,7
140 DATA 2,12,2,12,2,12,2,11,2
,9,2,11
150 DATA 4,12,2,7,4,7,2,7
160 DATA 2,12,2,12,2,12,2,11,2
,9,2,11
170 DATA 4,12,2,7,4,7,2,5
180 DATA 6,4,6,2,12,0
```

Ao rodar o programa, devemos responder, com um número entre 1 e 50, à solicitação do computador de um valor para o andamento. Como já foi explicado, trata-se do inverso do andamento. O programa nada mais faz do que multiplicar cada duração pelo valor do andamento — portanto, quanto maior o valor, menor a velocidade.

Usamos o inverso do andamento para tornar o programa mais simples e rápido. Um comando **IF...THEN** verifica se o andamento está na faixa permitida. A linha 30 executa a melodia. Um laço **FOR...NEXT** obtém com **READ**

5

Símbolo da nota	Nome da nota	Símbolo da pausa	Duração relativa	Duração quando pontuada
𝅝	Semibreve	𝄻	16	24
𝅗𝅥	Mínima	𝄼	8	12
♩	Semínima	𝄽	4	6
♪	Colcheia	𝄾	2	3
𝅘𝅥𝅯	Semicolcheia	𝄿	1	1.5



dois valores para cada nota: um para a tonalidade e outro para a duração. Um comando **SOUND** emite o som antes que outros dados sejam lidos.

Ao executar suas próprias canções, acerte os valores do laço **FOR...NEXT** para a leitura do número exato de notas. Por enquanto, o laço nunca termina, pois tem um **STEP** de valor 0.

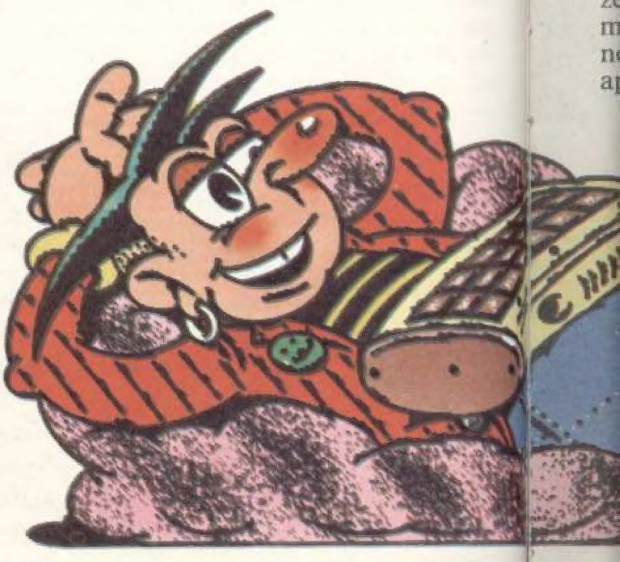
```
10 INPUT "ANDAMENTO (1-50)";TP
20 IF TP<1 THEN 10
30 PLAY "T"+STR$(INT(32+223/TP)
)
40 FOR I=1 TO 48
50 READ A$,D
60 PLAY "L"+STR$(INT(1+63/D))+"
03"+A$
70 NEXT I
1000 DATA E,6,D,6,C,12
1010 DATA E,6,D,6,C,12
1020 DATA G,6,F,4,F,2,E,12
1030 DATA G,6,F,4,F,2,E,10
1040 DATA G,2,04C,4,04C,2,B,2,A
,2,B,2,04C,4,G,2,G,4,G,2
1050 DATA 04C,2,04C,2,04C,2,B,2
,A,2,B,2,04C,4,G,2,G,2,G,2,G,2
1060 DATA 04C,4,04C,2,B,2,A,2,B
,2,04C,2,G,2,G,2,G,4,F,2
1070 DATA E,6,D,6,C,12
```

O programa do MSX começa pedindo que o usuário informe o andamento desejado. A linha 20 verifica se o valor escolhido é válido — se não for, o programa volta à linha 10. A linha 30 cuida de estabelecer o andamento, selecionando um valor adequado para o argumento "T" do comando **PLAY**. O valor escolhido no início deve ser transformado em cordão pela instrução **STR$**, para que possa servir como argumento de **PLAY**.

A linha 50 obtém os valores de tonalidade e duração das notas. A linha 60 produz o som, também usando o comando **PLAY**. Para determinar a duração de cada nota, **STR$** transforma em cordão o valor obtido nas linhas **DATA** e o coloca logo após "**L**". O "**03**" que se segue determina a oitava (eventualmente, é cancelado pelo "**04**" de uma nota). Finalmente, a variável A$ estabelece a altura da nota.

Para executar a música, usamos um laço **FOR...NEXT** (linhas 40 a 70). Evitamos, assim, a emissão de uma mensagem de erro que, além de deselegante, causa problemas no MSX. Como esse computador possui um microprocessador específico para a produção de sons, a música continua tocando, mesmo após a execução do programa. Havendo uma mensagem de erro, o sistema operacional do MSX suspende imediatamente todas as atividades periféricas em andamento — tela gráfica, motor do gravador e som. Assim, se deixássemos passar um erro, a música seria interrompida.

```
10  HOME
20  FOR I = 0 TO 22: READ A: PO
KE 800 + I,A: NEXT
30  DATA  160,0,174,133,3,238,4
8,192,136,208,5,206,132,3,240,6
,202,208,245,76,34,3,96
40  INPUT "ANDAMENTO (1-50)";T
50 T = T / 3.3
60  READ A,B
70  POKE 900,1 + B * T: POKE 90
1,A: CALL 800
80  FOR I = 1 TO 100: NEXT
90  GOTO 60
100  DATA  76,6,85,6,96,12
110  DATA  76,6,85,6,96,12
120  DATA  64,6,72,4,72,2,76,12
130  DATA  64,6,72,4,72,2,76,10
,64,2
140  DATA  47,4,47,2,50,2,56,2,
50,2
150  DATA  47,4,64,2,64,4,64,2
160  DATA  47,2,47,2,47,2,50,2,
50,2,50,2
```



```
170  DATA  47,4,64,2,64,2,64,2,
64,2
180  DATA   47,4,47,2,50,2,56,2
,50,2
190  DATA  47,2,64,2,64,2,64,4,
72,2
200  DATA  76,6,85,6,96,12
```

O programa do Apple usa a rotina em código explicada no artigo da página 706. As linhas 20 e 30 colocam essa rotina na memória, depois que o **HOME** da linha 10 limpou a tela.

A linha 40 solicita o valor desejado para o andamento da canção e a linha 50 corrige seu valor. O fator de correção aqui escolhido é compatível com durações relativas com valores até 16 (semibreve). Como em nossa canção não existem notas com duração tão longa, podemos alterar o valor para 2.5, se quisermos. Nesse caso, porém, a duração máxima de cada nota deverá ser 12 (mínima pontuada).

A linha 60 obtém os valores de tonalidade e duração e à linha 70 cabe executar a nota. Para chamarmos a rotina (**CALL 800**) que produz o som, o endereço 900 deve conter a duração e o 901, a tonalidade.

O problema de tempo, que mencionamos ao tratar dos demais computadores, não se aplica ao micro Apple. A linha 80 produz um pequeno atraso, fazendo com que a música não seja executada muito rapidamente.

A linha 90 manda o programa de volta à linha 60 para que o programa obtenha o valor de uma nova nota. O programa termina com uma mensagem de erro, informando o fim dos dados.

Embora o programa anterior funcione no TK-2000, seus usuários devem fazer algumas modificações. Primeiro, como a rotina de produção de sons não é necessária, as linhas 20 e 30 podem ser apagadas. Além disso, devemos substituir os comandos na linha 70 do programa do Apple pelo comando **SOUND**. A linha 80 também precisa ser modificada, já que o TK-2000 é mais lento que o Apple. Aqui estão as modificações:

```
70 SOUND A, 1+B*T
80 FOR I=1 TO 40:NEXT
```

```
10 INPUT"ANDAMENTO (1-50)";TP
20 IF TP<1 THEN 10
30 READ A$,D
40 PLAY"T"+STR$(INT(1+255/(TP*D
))))+"03"+A$
50 GOTO 30
1000 DATA E,6,D,6,C,12
1010 DATA E,6,D,6,C,12
1020 DATA G,6,F,4,F,2,E,12
1030 DATA G,6,F,4,F,2,E,12
1040 DATA O4C,4,O4C,2,B,2,A,2,B
,2,O4C,4,G,2,G,4,G,2
1050 DATA O4C,2,O4C,2,O4C,2,B,2
,A,2,B,2,O4C,2,G,2,G,2,G,2,G,2
1060 DATA O4C,4,O4C,2,B,2,A,2,B
,2,O4C,2,G,2,G,2,G,4,F,2
1070 DATA E,12,D,12,C,12
```

O programa do TRS-Color começa pedindo o valor da velocidade de execução desejada — ou andamento — por meio de um comando **INPUT**.

A linha 20 verifica se o valor fornecido está dentro dos limites válidos, retornando à linha 10 se o andamento for menor que 1.

A linha 30 lê dois valores para cada nota: um para a tonalidade — representada por meio de uma letra — e outro para a duração. Para simbolizar os sustenidos, usamos um sinal "cardinal" ou um "mais" antes da nota e, para os bemóis, um sinal "menos" no mesmo lugar. A linha 40 toca a nota, através de um comando **PLAY**. O **T** entre aspas faz com que o número que vem a seguir, entre 1 e 255, determine o andamento da melodia. Esse número é calculado invertendo-se o valor inicialmente fornecido ao programa. Para ser utilizado como argumento de **PLAY**, o número deve ser convertido em um cordão por intermédio do comando **STR$**.

O cordão "03" seleciona a oitava apropriada; a parte final da linha (+ **A$**) cuida da tonalidade da nota.

Após produzir o som de uma nota, o programa volta à linha 30, onde **READ** lê um novo par de dados nas linhas **DATA**. Isto se repete até que o conteúdo das linhas **DATA** se esgote. O programa é interrompido com uma mensagem de erro, ao tentar prosseguir a leitura de dados — que já acabaram.

UM VIDEOGAME EM ASSEMBLER

Começamos aqui a programação de um videogame completo: *Avalanche*. Em cada nova seção você encontrará uma surpresa... e aprenderá um pouquinho mais sobre código de máquina!

Os princípios mais importantes da linguagem de máquina podem ser ensinados por meio da programação de jogos. E, sem dúvida, aprender desse jeito é muito mais divertido. Desenvolvemos, assim, um videogame completo, especialmente construído para demonstrar os recursos de programação do Spectrum, do MSX e do TRS-Color. Não haverá programas para as linhas Apple, TK-2000, TRS-80 e ZX-81.

Avalanche é um videogame no estilo de *Keystone Kappers*, exigindo rapidez e agilidade do jogador, que precisa correr e saltar para escapar dos perigos que o ameaçam.

Serão criadas quatro telas, uma para cada nível de dificuldade — o jogo vai se tornando cada vez mais difícil. O personagem central de toda a ação é Willie. Nosso herói decide fazer um piquenique em uma montanha à beira-mar. Depois de encontrar um bom lugar para estender sua toalha, vai dar um passeio pelas redondezas. Ao voltar, descobre que alguns cabritos monteses espalharam seu lanche por toda a encosta. Para recuperar o que trouxe, precisará ir até o topo da montanha — o que deve fazer o mais rápido possível, levando-se em conta que a maré está subindo. Willie morrerá afogado se não chegar lá em cima a tempo.

Na primeira etapa do jogo, nosso herói é bombardeado por uma avalanche de pedras que rolam montanha abaixo. Para isso, utilizamos a primeira tela, que mostra as pedras descendo a encosta. Willie precisa saltá-las, se quiser continuar vivo — qualquer descuido pode ser fatal. A corrida e os saltos do personagem são controlados pelas teclas N e M. Se uma pedra atingi-lo, é morte certa. Felizmente, ele tem cinco vidas.

Quando Willie chega ao topo da montanha e recupera parte do que perdeu, é obrigado a voltar ao nível do mar. O jogo recomeça, então, na segunda tela. Desta vez, ao tentar subir o monte, encontrará pelo caminho enormes buracos, verdadeiras armadilhas cavadas na encosta. Se cair num deles ficará enterrado... e morto.

Chegando ao cume da segunda montanha, nosso herói será colocado aos pés da terceira e precisará enfrentar uma nova escalada. A encosta continua cheia de buracos, mas, agora, cada um deles é habitado por uma serpente venenosa, que tentará picar o personagem quando este saltar.

Na quarta tela, Willie terá que mostrar toda a sua habilidade, pois será bombardeado pela avalanche enquanto salta sobre os buracos habitados pelas serpentes "assassinas".

Nas quatro etapas do jogo a rapidez

- ESTRUTURA GERAL DO JOGO
- PROGRAMAS BASIC QUE AUXILIAM A MONTAGEM
- TABELAS DE DADOS EM ASCII
- CRIAÇÃO DE TELAS
- ROTINAS DA ROM
- IMPRESSÃO DE CARACTERES NA TELA
- PROGRAMAÇÃO DE PAUSAS
- CORREÇÃO DE ERROS

é fundamental, pois a maré está sempre subindo. Os pontos do jogador são contados conforme seu sucesso em escalar a montanha, havendo um bônus para quem conseguir passar as quatro telas sem perder nenhuma vida.

Além de ensinar os truques da programação em código de máquina, *Avalanche* é um jogo envolvente e cheio de emoções. Publicaremos suas seções em várias partes. O funcionamento de cada uma e seu papel na estrutura global do programa serão explicados detalhadamente na medida em que forem apresentadas. Sempre que possível, indicaremos as alternativas de utilização de determinadas rotinas. No final da série, você terá um videogame tão bom como os disponíveis em cartuchos.

A ESTRUTURA DO PROGRAMA

O cenário e os caracteres móveis são todos desenhados com blocos gráficos. Colocamos o cenário na tela por meio de laços que obtêm os caracteres correspondentes em bancos de blocos criados previamente na memória.

Os buracos e as cobras são superpostos ao cenário original, de maneira que não precisamos redesenhá-lo para montar as demais telas.

A parte principal do programa consiste em uma rotina de controle que determina o andamento e a prioridade dos eventos. Cada evento em si é executado por uma sub-rotina específica.

A animação do personagem central e das serpentes "assassinas" é feita em saltos de meio caractere, procedimento viabilizado pela criação de dois conjuntos de caracteres.

Com esse tipo de animação, pode-se obter suavidade nos movimentos sem complicar demais o programa ou atrasá-lo — o que comprometeria o desenvolvimento do jogo, já que a velocidade, no caso, é muito importante.

S

A primeira coisa que se programa em qualquer jogo é uma tela exibindo o título. Embora a rotina de impressão esteja em código de máquina, será mais fácil colocar as letras na memória usando um programa BASIC.

Antes de executar o programa, proteja a região da memória correspondente com **CLEAR 57434.**

```
10 LET X=57435
20 READ A$
30 FOR N=1 TO 38
40 POKE X,CODE A$(N)
50 LET X=X+1
60 NEXT N
70 DATA "AVALANCHECRIACAO:A.D
OEPROGRAMA:P.CLARK"
```

O programa cria na memória uma tabela com os códigos ASCII das letras que compõem a tela. Essa parte da memória deve ser gravada juntamente com o resto do programa em código; assim, na realidade, o programa BASIC não será necessário à execução do jogo. Grave-o, contudo, para auxiliar na montagem do programa completo.

Para que o Assembler de INPUT funcione a contento, faça as seguintes modificações: na linha 5120, troque o número 6 por 22; na linha 6110, adicione **:GOSUB 6150**; na linha 6150 troque **GOSUB 6260** por **GOSUB 6160**. Carregue, então, o Assembler e digite esta primeira rotina:

```
 40 REM org 58035
 50 REM ti call cl
 60 REM ld a,2
 70 REM out 254,a
 80 REM ld a,16
 90 REM ld (23624),a
100 REM ld ix,57435
110 REM ld b,9
120 REM ld a,70
130 REM ld hl,299
140 REM call me
150 REM ld b,15
160 REM ld a,7
170 REM ld hl,616
180 REM call me
190 REM ld b,18
200 REM ld hl,679
210 REM call me
220 REM ld b,2
230 REM ldp ld hl,65000
240 REM ld de,0
250 REM ldq dec hl
260 REM push hl
270 REM sbc hl,de
280 REM pop hl
290 REM jr nz,ldq
300 REM djnz ldp
301 REM nop
302 REM nop
303 REM nop
304 REM nop
305 REM nop
306 REM nop
307 REM nop
308 REM nop
310 REM ret
```

Grave os mnemônicos usando a opção correspondente do Assembler. Depois, monte a primeira rotina e grave o programa em código — se preciso, recorra ao monitor de linguagem de máquina. Digite, então, mais esta rotina:

```
10 REM org 58146
20 REM ktt ld a,253
30 REM in a,254
40 REM bit 1,a
50 REM jr nz,ktt
60 REM ret
70 REM me push bc
80 REM push af
90 REM ld a,(ix+0)
100 REM call asc
110 REM pop af
120 REM call print
130 REM inc hl
140 REM inc ix
150 REM pop bc
160 REM djnz me
170 REM ret
180 REM asc push hl
190 REM ld hl,15608
200 REM ld de,8
210 REM ld b,31
220 REM sub b
230 REM ash add hl,de
240 REM dec a
250 REM jr nz,ash
260 REM push hl
270 REM pop bc
280 REM pop hl
290 REM ret
300 REM cl ld ix,16384
310 REM ld hl,6912
320 REM ld a,0
330 REM clp ld (ix+0),a
340 REM inc ix
350 REM dec hl
360 REM push hl
370 REM ld de,0
380 REM sbc hl,de
390 REM pop hl
400 REM jr nz,clp
410 REM ret
420 REM print push af
430 REM push hl
440 REM push bc
450 REM push hl
460 REM pop de
470 REM ld a,d
480 REM cp l
490 REM jr c,next
500 REM push de
510 REM ld de,1792
520 REM add hl,de
530 REM pop de
540 REM ld a,d
550 REM cp l
560 REM jr z,next
570 REM push de
580 REM ld de,1792
590 REM add hl,de
600 REM pop de
610 REM next push de
620 REM ld de,16384
630 REM add hl,de
640 REM pop de
650 REM ld a,8
660 REM pop bc
670 REM push af
680 REM rept ld a,(bc)
690 REM ld (hl),a
700 REM inc h
710 REM inc bc
720 REM pop af
730 REM dec a
740 REM jr z,exit
```

```
750 REM push af
760 REM jr rept
770 REM exit pop hl
780 REM pop af
790 REM push de
800 REM ld de,22528
810 REM add hl,de
820 REM pop de
830 REM ld (hl),a
840 REM push de
850 REM pop hl
860 REM ret
```

Grave essas duas rotinas separadamente. O programa pode ser executado pelo comando **RAND USR 58035**, como de costume. Lembre-se, porém, de que a tabela com dados em ASCII — que começa em 57435 — deve estar na memória para que o programa funcione.

O PROGRAMA BASIC

O programa escrito em BASIC resume-se a um laço **FOR...NEXT** que coloca as letras das palavras da página inicial na memória, formando uma tabela ASCII que o programa em código pode utilizar. Ele apenas ajuda a montar o programa, não sendo necessário ao funcionamento do jogo depois que a tabela tiver sido gravada.

OS CÓDIGOS

O programa consiste em uma rotina principal que chama as demais rotinas. Podemos, assim, cuidar de cada rotina separadamente, o que facilita muito a detecção e correção de erros.

A primeira instrução **call cl** chama a sub-rotina de limpeza da tela. Os comandos **ld a,2** e **out 254,a** definem as cores da borda, como foi explicado na página 556. O uso do comando **out** modifica apenas temporariamente a cor da moldura. Para tornarmos essa alteração permanente, precisamos modificar o valor da variável do sistema correspondente, que fica na posição 23624.

A cor da moldura especificada no comando **out** é 2, ou vermelho. Mas, para que o 2 seja colocado em 23624, devemos "deslocá-lo" — operação **SHIFT** — três bits para a esquerda, resultando no valor 16.

IMPRESSÃO NA TELA

A rotina rotulada com **me** controla a impressão de caracteres na tela. Certos parâmetros devem ser fornecidos a ela para que possa imprimir o texto correto na posição desejada.

O comando **ld ix,57435** coloca no registro IX o endereço do primeiro byte da tabela ASCII, a fim de que a rotina de impressão saiba o que imprimir.

O acumulador contém o atributo do caractere que será impresso. Tanto em linguagem de máquina quanto em BASIC o funcionamento é o mesmo. Quando o bit 7 está ligado, o caractere é cintilante — **FLASH**. O bit 6 dá brilho ao caractere — **BRIGHT**. Os próximos três bits determinam a cor do fundo, e os três seguintes são responsáveis pela cor dos caracteres.

No programa-fonte, colocamos 70 — 01000110 em binário —, que tem o bit 6 ligado, os bits que controlam a cor do fundo nulos e os bits dos caracteres valendo 6. Teremos, assim, caracteres amarelos sobre fundo preto, brilhante e não cintilante.

O registro B funciona como contador de caracteres. O valor nele colocado determina o comprimento do texto que será impresso. Na primeira vez que a rotina **me** é chamada, B contém 9 — as nove letras de *AVALANCHE*.

O registro HL contém a posição da tela a partir da qual o texto deve ser impresso. Essa posição é calculada em números de posições a partir do canto esquerdo da tela. Quando HL contém 139, o primeiro caractere da primeira cadeia de caracteres — ou seja, o "A" de *AVALANCHE* — é colocado na quinta linha da tela, onze posições a partir do lado esquerdo.

A rotina **me** é chamada três vezes para imprimir cada uma das três linhas da tela inicial.

PROGRAMAÇÃO DE PAUSAS

Para que o jogador possa ler o título, devemos incluir uma pausa no programa. Colocamos 2 no registro B, de modo que o laço responsável pela pausa seja executado duas vezes.

O número 65000 é colocado no registro HL. O conteúdo desse par diminui em uma unidade a cada volta do laço interno — rotulado com **ldq**. HL é guardado e recuperado da pilha somente para ganharmos tempo.

Pode parecer estranho subtrair zero — conteúdo de DE — de HL a cada volta do laço. Mas trata-se apenas de um recurso utilizado para afetar o indicador de zeros, que não reage à instrução **pop**. O comando **jr nz** depende desse indicador para que o programa possa sair do laço. Quando HL for reduzido a zero e **sub hl, de** executar a próxima subtração, o resultado obtido será zero; ativado o indicador de zeros, o processador sairá do laço.

Quando completo, o programa pros-

seguirá imprimindo a página de instruções. Mas, por enquanto, a instrução **ret** provoca o retorno ao BASIC.

Não se importe com os comandos **nop** que aparecem no final do programa. Eles não têm nenhum efeito sobre o microprocessador e sua função aqui é somente preencher espaço.

ROTINAS AUXILIARES

Algumas rotinas de apoio ao programa principal foram montadas. A que tem o rótulo **ktt** aguarda que pressionemos uma certa tecla, e ainda não será usada. O princípio de verificação do teclado aqui utilizado é igual ao explicado no artigo da página 381.

A rotina **me**, encarregada de controlar a impressão na tela, começa guardando os conteúdos de BC e AF na pilha. Em seguida, o conteúdo da posição de memória apontada pelo registro IX (que contém o endereço inicial da tabela ASCII) é colocado no acumulador por **ld a,(ix+0)**, comando que emprega o endereçamento indexado.

Os registros IX e IY são chamados de registros-índices, e dispõem de dezesseis bits para acomodar um endereço de memória. A sintaxe desse tipo de comando — muito utilizado, aliás, quando trabalhamos com tabelas — é **ix+d** ou **iy+d**, onde **d** é um valor entre 0 e 256 que pode ser somado ao endereço apontado pelo registro-índice.

Depois de colocar em A o código do caractere que deve ser impresso na tela, o programa chama a sub-rotina **asc**, que calcula a posição do padrão do caractere dentro de uma tabela de padrões existente na ROM (veja o artigo da página 381). O endereço inicial do padrão do caractere está em BC, quando o processador retorna da sub-rotina. Recupera-se o valor de AF da pilha e uma nova sub-rotina — **lprint**, que cuida da impressão propriamente dita — é chamada. Quando o processador retorna de **lprint**, HL contém a posição da tela onde houve impressão — na realidade, a posição do byte superior do padrão do caractere. A instrução **inc hl** aumenta este valor em uma unidade. O índice IX também é aumentado, passando a apontar para a próxima letra da tabela ASCII.

O par BC (B continha originalmente o comprimento do texto a ser impresso) é recuperado da pilha; enquanto seu conteúdo não for zero, a instrução **djnz me** diminui B em uma unidade e repete o processo de impressão.

A rotina **asc** calcula a posição do padrão do caractere a ser impresso. Em primeiro lugar, ela coloca temporariamente na pilha, por **push hl**, a posição de impressão que está em HL, pois vamos usar esse par de registros. Nele é colocado o endereço inicial da tabela de padrões — 15608. O número 8 é, então, colocado em DE e o 31, em B. A instrução **sub b** subtrai 31 de A. A operação é necessária porque a tabela de padrões começa com o caractere de código 32 — espaço em branco. Os códigos com valores menores que 32 são comandos BASIC e não caracteres.

Um caractere é formado por oito bytes, cada qual correspondendo ao padrão de uma das linhas que compõem a letra. Assim, cada oito bytes consecutivos na tabela de padrões correspondem a um caractere. O laço com o rótulo **ash** vai pulando de oito em oito bytes, até encontrar a posição do caractere que queremos imprimir. Para isso, utiliza A como contador. Em outras palavras, **add hl,de** soma 8 a HL e **dec a** subtrai um de A. A instrução **jr nz, ash** repete o processo até que A se reduza a zero. O endereço do byte inicial do padrão desejado estará, então, em HL. Este valor é transferido para a pilha e, em seguida, recuperado em BC. O valor original de HL — posição na tela — também é recuperado da pilha e a sub-rotina termina.

A sub-rotina **lprint** cuida da impressão do caractere. Ela começa guardando na pilha os pares AF, HL e BC. A continha o atributo do caractere, HL, a posição de impressão na tela, e BC, o endereço inicial do padrão do caractere. As linhas seguintes realizam uma série de testes com a posição de impressão. Ao final deles, DE contém a posição de impressão na tela.

O programa continua no rótulo **next**. Ali, **push de** coloca a posição de impressão na pilha e **ld de,16384** coloca em DE o endereço inicial da tela. O endereço da RAM em que vamos colocar o padrão do caractere é a soma da posição de impressão com o endereço inicial da tela — **add hl,de**. O resultado fica em HL. Recupera-se a posição de impressão em DE e o acumulador recebe o valor 8. BC recupera o endereço inicial do padrão e AF é colocado na pilha, onde A servirá de contador.

O rótulo **rept** representa o laço que desenha a letra. A instrução **ld a,(bc)** coloca em A o byte que contém uma linha do caractere e **ld (hl),a** transfere este mesmo valor para a tela. BC aponta para a tabela de padrões, enquanto HL aponta para a tela. A instrução **inc h** soma uma unidade ao registro H, o que equivale a somar 256 a HL. O valor resultante aponta para o byte imediatamente inferior ao anterior, de maneira que o caractere é desenhado linha por linha.

BC também tem seu conteúdo aumentado em uma unidade por **inc bc**. O contador do laço é recuperado da pilha — **pop af** — e A é diminuído em uma unidade. Se A não for igual a zero, a instrução **jr z,exit** é ignorada, AF volta para a pilha e o processador retorna para desenhar mais uma linha do caractere.

Como A continha inicialmente o valor 8, serão desenhadas exatamente oito linhas, uma abaixo da outra, completando uma letra. Quando A contém zero, **jr z,exit** desvia o programa para o rótulo **exit**. O atributo do caractere é, então, colocado na tabela de atributos. Esse parâmetro determina a cor do caractere. Recupera-se a posição de impressão em HL e o atributo vai para A — **pop hl** e **pop af**.

O conteúdo de DE volta para a pilha e este par recebe o endereço inicial da tabela de atributos — 22528. A instrução **add hl,de** calcula o endereço da RAM correspondente e o atributo é colocado nele por **ld (hl),a**. A posição de impressão na tela, preservada ora em DE, ora na pilha, volta para HL e a sub-rotina termina.

A sub-rotina **cl** limpa a tela, sendo chamada diversas vezes no decorrer do programa. Primeiro, o registro IX recebe o endereço inicial da tela — 16384. HL, por sua vez, recebe o número de bytes da mesma — 6912 —, servindo como contador. A instrução **ld a,0** coloca 0 em A. A instrução **ld (ix+0),a** coloca

0 na posição da tela apontada por IX, o que, efetivamente, limpa aquele byte. O registro IX, aumentado em uma unidade, passa a apontar para o próximo byte da tela. HL é diminuído em uma unidade e, enquanto não chegar ao valor zero, a instrução **jr nz,clp** retornará para apagar o próximo byte da tela. Na realidade, existem quatro linhas entre **dec hl** e **jr nz,clp**; contudo, elas só têm a função de prolongar o tempo, como o rótulo **ldq** da rotina anterior.

A primeira coisa a programar é uma página anunciando o videogame. As linhas seguintes criam essa página, obtendo os dados necessários em uma tabela, que montaremos mais tarde. Por enquanto, carregue o Assembler de INPUT com as seguintes modificações, para que funcione de maneira satisfatória: na linha 5120, troque o número 6 por 22; na linha 6110, adicione **:GOSUB 6150**; na linha 6150, troque **GOSUB 6260** por **GOSUB 6160**. Modifique a linha 5000 com **CLEAR 1500,&CFFF**, para proteger o topo da memória. Digite, em seguida, este programa:

```
10    org -12288
20    call 108
30    ld de,296
40    ld hl,-14000
50    ld bc,9
60    call 92
70    ld de,493
80    ld hl,-13991
90    ld bc,15
100    call 92
110    ld de,572
120    ld hl,-13976
130    ld bc,18
140    call 92
150    ld b,10
160    ldp ld hl, 65000
170    ld de,0
180    ldq dec hl
190    push hl
200    sbc hl,de
210    pop hl
220    jr nz,ldq
230    djnz ldp
240    ret
250    end
```

Grave o programa-fonte e depois monte-o. O programa em código pode ser gravado depois disso.

OS CÓDIGOS

O programa liga a tela de texto de quarenta colunas por intermédio de uma rotina da ROM cujo endereço é 108, em decimal. Em seguida, reserva uma porção da memória para os códigos ASCII do texto a ser impresso. Ela começa no endereço -14000.

Uma outra rotina da ROM, que fica no endereço 92, encarrega-se da transferência de uma área da RAM para a VRAM. Para que o texto certo seja colocado na posição desejada, certos parâmetros devem ser fornecidos.

O par DE precisa conter a posição inicial da área da VRAM em que vamos colocar o texto. O par HL, por sua vez, contém o endereço inicial da área da RAM que será transferida. BC, finalmente, funciona como contador de bytes, determinando que comprimento terá o texto que vai ser impresso.

Essa rotina da ROM é chamada três vezes, uma para cada linha que aparece na tela inicial. Antes de cada comando **call**, os valores adequados são colocados nos pares DE, HL e BC.

PROGRAMAÇÃO DE PAUSAS

Para que o jogador possa ler o título, devemos incluir uma pausa no programa. Colocamos 10 no registro B, de modo que o laço responsável pela pausa seja executado dez vezes.

O número 65000 é colocado em HL, cujo conteúdo diminui em uma unidade a cada volta do laço interno — rotulado com **ldq**. HL é guardado e recuperado da pilha só para ganharmos tempo.

Pode parecer estranho subtrair zero — conteúdo de DE — de HL a cada volta do laço. Mas trata-se apenas de um recurso utilizado para afetar o indicador de zeros, que não reage à instrução **pop**. O comando **jr nz** depende desse indicador para que o programa possa sair do laço. Quando HL for reduzido a zero e **sub hl, de** subtrair zero dele, o resultado será igual a zero; ativado o indicador de zeros, o processador sairá do laço.

Quando o programa estiver completo, ele prosseguirá imprimindo a página destinada à exibição das instruções. Mas, por enquanto, a instrução **ret** provoca o retorno ao BASIC.

O PROGRAMA BASIC

As linhas seguintes geram a tabela ASCII necessária ao funcionamento do programa em código.

```
5 CLS:CLEAR 100,-14000
10 A$="AVALANCHECriação: A. Doe
Programa: R. Neder"
20 FOR I=0 TO 41
30 POKE -14000+I,ASC(MID$(A$,I+
1,1))
40 NEXT
50 DEFUSR1=-12288
60 X=USR1(0)
70 CLS
80 END
```

A linha 5 limpa a tela e protege o topo da memória que conterá a tabela.

A linha 10 coloca o texto utilizado na página inicial dentro da variável **A$**. É, sem dúvida, bem melhor digitar uma tabela na forma de um texto do que digitar os números de todos os códigos ASCII correspondentes.

Em seguida, um laço **FOR...NEXT** coloca os códigos ASCII na memória, criando a tabela ASCII.

O próprio programa testa, então, a rotina em código — se ela estiver gravada na memória.

O jogo não vai precisar desse programa BASIC depois que a tabela tiver sido gravada. Guarde-o, porém, para ajudar na montagem.

T

A primeira coisa que se programa em qualquer jogo é uma tela exibindo o título. Embora a rotina de impressão esteja em código, será mais fácil colocar os códigos ASCII das letras na memória usando um programa BASIC:

```
1 CLEAR 200,16999
10 AD=17000
30 READ A$
40 FOR A=1 TO LEN(A$):B=ASC(MID
$(A$,A,1))
50 IF B<&H61 THEN POKE AD,B ELS
E POKE AD,B-96
60 AD=AD+1
70 NEXT A
80 DATA "avalanchecriacao: a.do
e programa: s. kellaway e g. he
dley"
```

Quando ele é executado, a tabela necessária ao jogo é criada na memória do microcomputador. Para gravar a tabela, utilize **CSAVEM "DADOS", 17000, 17059, 19000**.

Se quiser usar o Assembler de INPUT, precisará ampliar a memória disponível; se não o fizer, o espaço será insuficiente para o Assembler e o programa em código. Digite, então:

```
POKE 25,6
POKE 26,1
NEW
```

Depois, digite **CLEAR 200,18999** para proteger o topo da memória e, em seguida, o programa:

```
10  ORG 19000
20  JSR CLS
30  LDX #1066
40  LDY #17000
50  LDB #9
60  JSR LPRINT
70  LDX #1384
80  LDB #15
90  JSR LPRINT
100  LDX #1445
110  LDB #21
120  JSR LPRINT
130  LDX #1481
140  LDB #12
150  JSR LPRINT
160  LDA #5
170  PAUSE LDX #65535
180  PAUSEI LEAX -1,X
190  BNE PAUSEI
200  DECA
210  BNE PAUSE
220  JSR CLS
230  RTS
240  LPRINT EQU 19174
250  CLS EQU 19148
```

Os mnemônicos podem ser gravados por meio da opção correspondente do Assembler. Monte o programa e grave-o em código — se não souber fazer isto, utilize o monitor Assembler.

Além desse programa e da tabela, as outras rotinas aqui apresentadas serão necessárias para que você veja o resultado por meio de **EXEC 19000**.

O PROGRAMA BASIC

O programa em BASIC começa reservando uma boa quantidade de memória para armazenar o programa em código. Em seguida, as letras utilizadas na primeira tela são colocadas na memória, representadas por seus códigos ASCII. Em geral, é preciso subtrair 96 de cada código para que eles correspondam aos códigos de letras invertidas. Algumas letras, porém — códigos menores que 61, em hexadecimal —, não precisam dessa operação.

LIMPEZA DA TELA

A rotina em código começa em 19000, depois da tabela ASCII, e salta para a sub-rotina **CLS**, que começa em 19148. Na primeira rotina, o endereço desse rótulo é definido por uma instrução **EQU**, para que os desvios sejam calculados. Eis a sub-rotina:

```
10  ORG 19148
20  CLS LDX #1024
30  LDA #128
40  CLSI STA ,X+
50  CMPX #1536
60  BLO CLSI
70  RTS
```

O endereço inicial da tela — 1024 — é colocado no registro X e o código do espaço em branco — 128 —, em A.

STA,X + coloca esse código na posição apontada por X; o conteúdo de X aumenta em uma unidade. A rotina é repetida várias vezes até que X seja maior que 1536, endereço final da tela.

Como o conteúdo de X aumenta antes da colocação de 128 em um endereço da tela, um comando **BLO** — desvio se for "menor que" — é usado para sair do laço **CLSI**. Essa rotina deve ser gravada separadamente.

IMPRESSÃO NA TELA

Quando o microprocessador retorna da sub-rotina **CLS**, a rotina principal prepara os registros para que se faça a impressão na tela.

O registro X contém a posição que desejamos para o primeiro caractere da linha a ser impressa. O "A" de *Avalanche* será impresso, então, em 1066.

O registro Y contém a posição de memória da tabela ASCII da primeira letra a ser impressa. Assim, 17000 é colocado em Y. O registro B contém o número de caracteres a ser impresso. Como vamos começar escrevendo "AVALANCHE", 9 é colocado em B. Em seguida, ocorre um salto para a sub-rotina **LPRINT**. Mais uma vez uma instrução **EQU** define o endereço 19174 para efeito de cálculos de desvios.

A ROTINA LPRINT

Esta rotina começa em 19174. Sua função consiste em obter os dados da tabela ASCII, colocando os códigos na tela, um de cada vez.

```
10  ORG 19174
20  LPRINT LDA ,Y+
30  STA ,X+
40  DECB
50  BNE LPRINT
60  RTS
```

Os códigos da tabela apontados por Y são colocados em A e o apontador volta-se para a próxima posição. O comando **STA,X +** coloca-os, então, na posição de tela apontada por X, e X aumenta em uma unidade, apontando para a posição seguinte.

A instrução **DECB** subtrai uma unidade do conteúdo de B e a rotina é repetida até que B chegue a zero. Quando isso ocorre, o microprocessador retorna à rotina principal. Guarde esta rotina separadamente.

A rotina prossegue até completar a página de rosto de nosso videogame, imprimindo linha por linha.

Para escrever uma nova linha, a posição de impressão é colocada em X e o comprimento do texto, em B.

Não há necessidade de colocarmos um novo valor em Y, já que esse apontador simplesmente percorre a tabela ASCII B caracteres de cada vez.

A ROTINA DE PAUSA

Quando a última linha da página inicial tiver sido impressa, o microprocessador fará uma pausa no programa, para que possamos lê-la, antes de ser substituída pela tela de instruções.

O número 5 é colocado no acumulador e o registro X recebe o valor 65535. **LEAX -1,X** subtrai uma unidade de X e **BNE PAUSEI** retorna, de modo que vai diminuindo até ser reduzido a zero; A também é diminuído. O processo se repete até qué A também seja igual a zero e o comando **BNE** permita que o processador deixe o laço.

DOIS LAÇOS

O laço externo **PAUSE** — baseado no acumulador — repete-se, assim, cinco vezes; a cada repetição, o laço interno **PAUSEI** é executado 65535 vezes. O uso de dois laços tem a vantagem de possibilitar o acerto do tempo. O laço externo nos dá um ajuste grosseiro e o laço interno, um ajuste mais fino.

Por fim, a rotina **CLS** limpa a tela novamente. Em seguida, o computador passa à página de instruções, que apresentaremos no próximo artigo desta série. Por hora, uma instrução **RTS** retorna ao BASIC.

O JOGO DO OTELO (1)

Embora suas regras sejam bastante simples, o jogo do Otelo requer muita estratégia. Programá-lo é fácil: monte o tabuleiro, crie as peças e desafie sua máquina!

Otelo é um jogo de estratégia que se desenvolve num tabuleiro de oito por oito posições — como os tabuleiros de xadrez ou dama. As regras são bem fáceis e o jogo também.

O objetivo é tomar o maior número possível de peças do oponente. A jogada consiste basicamente na colocação das peças no tabuleiro, uma a uma, por um jogador de cada vez, até preenchê-lo totalmente. Cada jogador começa com duas peças e tenta tomar as do oponente, "cercando-as". Para consegui-lo, deve colocar as peças de maneira que as do oponente fiquem entre as suas — ou seja, que se forme uma trilha de peças adversárias com peças suas nas duas extremidades. Ao tomar uma trilha adversária, esteja ela na horizontal, vertical ou diagonal, todas as peças do oponente serão substituídas por peças do jogador em questão.

O placar de cada jogador corresponde ao número de peças que ele tem no tabuleiro naquele momento. Será vencedor aquele que tiver o maior número de peças no tabuleiro quando ele estiver totalmente cheio.

Nesta versão para computador, estaremos jogando contra a máquina que, além de mostrar o tabuleiro, também atualiza o placar.

DICAS PARA VENCER

Para quem nunca jogou Otelo, as dicas que se seguem serão muito úteis.

As posições dos cantos são extremamente importantes, já que, uma vez conquistadas, jamais serão roubadas. Em conseqüência, a ocupação de tais posições é fundamental para garantir a vitória — mesmo que implique o sacrifício de algumas de nossas peças ou impeça um outro movimento que pudesse nos render mais peças do oponente.

Se considerarmos que uma peça pode completar mais de uma linha — na vertical, na horizontal e na diagonal —, concluiremos que o movimento aparentemente mais óbvio nem sempre é o melhor, pois, nos estágios mais avançados do jogo, será possível tomar duas ou três linhas de peças do oponente com a simples adição de uma peça.

Pense sempre adiante. Podemos induzir o adversário a criar situações que nos ajudem a tomar posições vitais, simplesmente simulando ou fingindo que fizemos uma má jogada.

O PROGRAMA

Uma das grandes vantagens da versão de Otelo para computador sobre o verdadeiro tabuleiro é que nela o computador fica com todo o "trabalho pesado". Além de desempenhar o papel de adversário, ele substitui as peças conquistadas e cuida do placar, deixando-nos livres para pensar só nas jogadas.

O computador demora um pouco para fazer suas jogadas, pois leva em conta todos os movimentos possíveis. No decorrer do jogo, à medida que o número de posições vazias diminui, passa a tomar decisões com maior rapidez.

JOGANDO

Ao rodar o programa, o computador perguntará se você deseja fazer a primeira jogada ou não. Para executar uma jogada, é necessário fornecer-lhe duas coordenadas. Estas especificam a posição da peça no tabuleiro e estão compreendidas entre os valores 1 e 8. Fornecemos primeiramente a coordenada da linha — ou seja, do eixo Y, com origem no topo da tela — e, depois, a coordenada da coluna — eixo X, com origem no lado esquerdo do tabuleiro. Se fornecermos 0 (zero) para as coordenadas, o jogo será encerrado; se fornecermos 9, transferiremos o direito de jogar para o computador.

PRIMEIRAS LINHAS

Digite agora a primeira parte do jogo do Otelo. Se você rodar o programa como está, verá o tabuleiro na tela, mas ainda não poderá jogar.

Salve as linhas digitadas em disco ou em fita cassete para mais tarde incorporá-las à segunda parte do programa, que será apresentada num próximo artigo, completando o jogo.

```
10 BORDER 0: PAPER 7: INK 1:
CLS
15 PRINT AT 1,11; INVERSE 1;"
OTHELLO"
20 PRINT AT 10,2;"VOCE QUER C
OMECAR ?  (S OU N)": INPUT X$
: IF X$="" THEN  GOTO 20
30 LET X$=X$(1): IF X$<>"S"
AND X$<>"N" AND X$<>"s" AND X
$<>"n" THEN  GOTO 20
40 LET CP=1: IF X$="N" OR X$=
"n" THEN  LET CP=2
100 DIM B(8,8): DIM C(8): DIM
D(8,2): DIM X(60): DIM Y(60):
DIM N(60)
110 LET B(4,4)=1: LET B(4,5)=2
: LET B(5,4)=2: LET B(5,5)=1
120 FOR F=1 TO 8: READ A: LET
D(F,1)=A: READ A: LET D(F,2)=A
: NEXT F
130 DATA -1,-1,0,-1,1,-1,-1,0,
1,0,-1,1,0,1,1,1
140 FOR F=0 TO 7: READ A,B,C:
POKE USR "A"+F,A: POKE USR "B"
+F,B: POKE USR "C"+F,C: NEXT F
150 DATA 204,0,0,51,60,60,204,
126,66,51,126,66,204,126,66,51
,126,66,204,60,60,51,0,0
```

```
10  HOME : VTAB (3): HTAB (15)
15  INVERSE : PRINT " O T E L O
 ": NORMAL
20  VTAB (10): PRINT "QUER JOGA
R PRIMEIRO (S/N) ";: INPUT X$:
IF X$ = "" THEN 20
30 X$ =  LEFT$ (X$,1):CP = 1: I
F X$ = "N" THEN CP = 2
40  IF X$ <  > "S" AND X$ <  >
"N" THEN  GOTO 20
100  DIM B(8,8),C(8),D(8,2),X(6
0),Y(60),N(60)
110 B(4,4) = 1:B(4,5) = 2:B(5,4
) = 2:B(5,5) = 1
120  FOR F = 1 TO 8: READ A:D(F
,1) = A: READ A:D(F,2) = A: NEX
T
130  DATA  -1,-1,0,-1,1,-1,-1,0
,1,0,-1,1,0,1,1,1
```

```
5 COLOR 1,15,15:SCREEN2
10 FOR X=1 TO 16:READ A:NEXTX:G
OSUB 9200:RESTORE
```

- AS REGRAS DO JOGO
- DICAS PARA VENCER
- A PRIMEIRA TELA
- SELEÇÃO DOS GRÁFICOS PARA O TABULEIRO
- E AS PEÇAS
- INICIALIZAÇÃO DO JOGO
- DIGITANDO A ROTINA PRINCIPAL
- A VEZ DO JOGADOR


```
15 DRAW"BM60,40;S25;C1":A$="
LO":GOSUB 9300
20 DRAW"BM26,120;S10;C2":A$="QU
ER JOGAR PRIMEIRO":GOSUB 9300:D
RAW"BM90,140":A$="S OU N":GOSUB
 9300
21 X$=INKEY$:IFX$=""THEN 21
25 COLOR15,4,4
30 IFX$<>"S"ANDX$<>"s"ANDX$<>"N
"ANDX$<>"n"THEN 21
40 CP=1:IFX$="N"ORX$="n"THEN CP
=2
100 DIM B(8,8),C(8),D(8,2),X(60
),Y(60),N(60)
110 B(4,4)=1:B(4,5)=2:B(5,4)=2:
B(5,5)=1
120 FOR F=1 TO 8:READ A:D(F,1)=
A:READ A:D(F,2)=A:NEXTF
130 DATA -1,-1,0,-1,1,-1,-1,0,1
,0,-1,1,0,1,1,1
150 GOSUB 1100
```

```
10 PMODE 1,1:COLOR 4,1:PCLS:SCR
EEN 1,0:FOR X=1 TO 16:READ A:NE
XTX:GOSUB 9200:RESTORE
15 DRAW"BM60,40;S16":A$="OTHELL
O":GOSUB 9300
20 DRAW"BM26,120;S8;C2":A$="  Q
UER JOGAR PRIMEIRO":GOSUB 9300:
DRAW"BM100,140":A$="S OU N":GOS
UB 9300
21 X$=INKEY$:IF X$="" THEN 21
30 IF X$<>"S" AND X$<>"N" THEN
21
40 CP=1:IF X$="N" THEN CP=2
100 DIM B(8,8),C(8),D(8,2),X(60
),Y(60),N(60)
110 B(4,4)=1:B(4,5)=2:B(5,4)=2:
B(5,5)=1
120 FOR F=1 TO 8:READ A:D(F,1)=
A:READ A:D(F,2)=A:NEXT F
130 DATA -1,-1,0,-1,1,-1,-1,0,1
,0,-1,1,0,1,1,1
150 GOSUB 1100
```

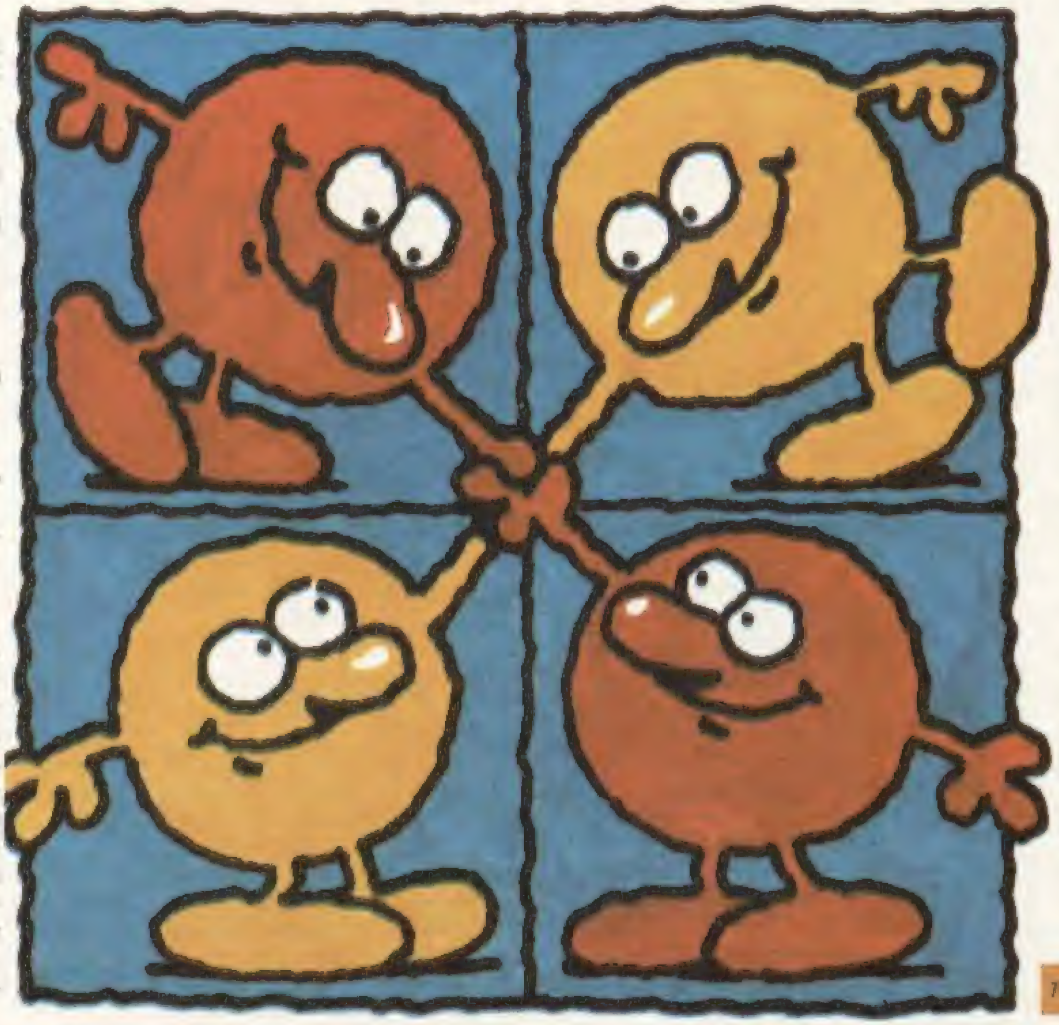
As linhas 10 a 40 geram a primeira tela que o jogador vê, ou seja, o título do jogo e a pergunta que o inicia. A linha 10 (linha 5, no MSX) se encarrega do ajustamento das cores e da limpeza da tela. O nome do jogo é gerado na linha 15. Como nos programas para os micros TRS-Color e MSX usou-se a tela de alta resolução, existe uma rotina que desenha letras no final do programa (a qual já utilizamos anteriormente em INPUT).

A linha 20 apresenta a primeira pergunta, destinada a definir quem jogará primeiro. A resposta, armazenada em **X$**, é depois reduzida à sua primeira letra, na linha 30. **CP** corresponde ao jogador em questão e recebe o valor 1 quando quem joga é você e 2 quando é o computador. A linha 40 faz a verificação por S e N.

As linhas 100 a 130 inicializam as variáveis e matrizes usadas no jogo. A linha 100 cuida das matrizes. **B(x,y)** representa o tabuleiro; os valores armazenados em cada elemento representam o status do quadrado correspondente no tabuleiro. Se o valor for 0, o quadrado está vazio; se for 1, uma peça pertencente ao jogador ocupa aquela posição; e se for 2, a posição está ocupada por uma peça do computador. **C(x)** é usado para verificar as jogadas do jogador. **D(x,y)** contém os deslocamentos X e Y nas oito direções possíveis da jogada. **X(x)**, **Y(x)** e **N(x)** são usadas nos cálculos para a jogada do computador.

A linha 110 especifica as posições iniciais no tabuleiro. Cada jogador tem duas peças posicionadas em seu centro, no começo da disputa. As direções possíveis, partindo dessa posição, são definidas pela linha **DATA** 130. Cada valor representa um deslocamento X ou Y, com os números negativos indicando esquerda ou topo do tabuleiro, respectivamente.

Na versão para os microcomputadores Spectrum, as linhas 140 e 150 geram UDG para o quadrado vazio e para as peças. A linha 140 procede à leitura dos dados apresentados pela linha 150 e cria os UDG **A**, **B** e **C**.



O LAÇO PRINCIPAL

```
500 GOSUB 1000
505 IF CS+PS=64 THEN  GOTO
 4000
510 LET EG=0: IF CP=1 THEN
GOSUB 2000: GOSUB 1000: IF EG=
1 THEN  GOTO 4000
515 IF CS+PS=64 THEN  GOTO
 4000
520 IF CP=2 THEN  GOSUB 3000
530 GOTO 500
```

```
500  GOSUB 1000
505  IF CS + PS = 64 THEN 4000
510 EG = 0: IF CP = 1 THEN  GOS
UB 2000: GOSUB 1000: IF EG = 1
THEN 4000
515  IF CS + PS = 64 THEN 4000
520  IF CP = 2 THEN  GOSUB 3000
530  GOTO 500
```

```
500 GOSUB 1000
505 IF CS+PS=64 THEN 4000
510 EG=0:IF CP=1 THEN GOSUB 200
0:GOSUB 1000:LINE(112,150)-(144
,180),1,BF:IF EG=1 THEN 4000
515 IF CS+PS=64 THEN 4000
520 IF CP=2 THEN GOSUB 3000
530 GOTO 500
```

```
500 GOSUB 1000
505 IF CS+PS=64 THEN 4000
510 EG=0:IF CP=1 THEN GOSUB 200
0:GOSUB 1000:COLOR 1:LINE(118,1
50)-(150,180),PSET,BF:IF EG=1 T
HEN 4000
515 IF CS+PS=64 THEN 4000
520 IF CP=2 THEN GOSUB 3000
530 GOTO 500
```

O laço principal do programa está entre as linhas 500 e 530. A linha 500 chama a sub-rotina que desenha o tabuleiro. As linhas 505 e 515 verificam se o tabuleiro está cheio, ou seja, se a soma dos pontos do jogador, **PS**, com os do computador, **CS**, resulta em 64 (número de posições do tabuleiro).

A linha 510 ajusta o indicador de final de jogo **(EG)** para 0 e chama a rotina responsável pela vez do jogador, seguida pela sub-rotina do tabuleiro. Se depois da execução dessas sub-rotinas o indicador **EG** for setado (valor 1), o programa saltará para a sub-rotina de final de jogo, que começa na linha 4000. A jogada do computador é feita pela linha 520, caso **CP** seja igual a 2 (sub-rotina 3000).

DESENHANDO O TABULEIRO

```
1000 CLS : PRINT TAB 11;"123456
78": LET PS=0: LET CS=0
1010 FOR F=1 TO 8: PRINT TAB 9;
F;" ";: FOR G=1 TO 8
1020 IF B(F,G)=0 THEN  PRINT CH
R$ 144;
1030 IF B(F,G)=1 THEN  PRINT  I
NK 2;CHR$ 145;: LET PS=PS+1
1040 IF B(F,G)=2 THEN  PRINT  I
NK 2;CHR$ 146;: LET CS=CS+1
1050 NEXT G: PRINT : NEXT F
1052 LET P$="PONTOS": IF PS=1 T
HEN  LET P$="PONTO"
1054 LET Q$="PONTOS": IF CS=1 T
HEN  LET Q$="PONTO"
1060 PRINT '' INK 2;" VOCE =";T
AB 20;"COMPUTADOR=": PRINT " ";
PS;" ";PS;TAB 20;CS;" ";Q$
1070 RETURN
```

```
1000  HOME : PRINT  TAB( 18);"1
2345678":PS = 0:CS = 0: PRINT
1010  FOR F = 1 TO 8: PRINT  TA
B( 16);F;" ";: FOR G = 1 TO 8
1015  IF B(F,G) <  > 1 AND B(F,
G) <  > 2 THEN B(F,G) = 0
1020  IF B(F,G) = 0 THEN  PRINT
 "+";
1030  IF B(F,G) = 1 THEN  PRINT
  CHR$ (48);:PS = PS + 1
1040  IF B(F,G) = 2 THEN  INVER
SE : PRINT " ";: NORMAL :CS = C
S + 1
1050  NEXT G: PRINT : NEXT F
1052 P1$ = " PONTOS": IF PS = 1
 THEN P1$ = " PONTO"
1054 P2$ = " PONTOS": IF CS = 1
 THEN P2$ = " PONTO"
1060  PRINT : PRINT "JOGADOR =
0"; TAB( 25);"COMPUTADOR = ";:
INVERSE : PRINT " ": NORMAL : P
RINT PS;P1$; TAB( 25);CS;P2$
1070  RETURN
```

```
1000 PS=0:CS=0
1010 FOR F=1TO8:FOR G=1TO8
1030 IF B(F,G)=1 THEN LINE((G-1
)*16+64,(F-1)*16+12)-(G*16+64,F
*16+12),15,BF:PS=PS+1
1040 IF B(F,G)=2 THEN LINE((G-1
)*16+64,(F-1)*16+12)-(G*16+64,F
*16+12),1,BF:CS=CS+1
1050 NEXTG,F
1060 DRAW"BM10,90;C1":A$="VOCE"
:GOSUB9300:DRAW"BM200,90":A$="C
OMPUT":GOSUB9300
1063 LINE(20,110)-(28,118),15,B
F:LINE(220,110)-(228,118),1,BF
1065 LINE(5,60)-(42,80),1,BF:LI
NE(205,60)-(242,80),1,BF
1066 DRAW"BM12,62;S16;C15":A$=M
ID$(STR$(PS),2):GOSUB9300:DRAW"
BM212,62":A$=MID$(STR$(CS),2):G
OSUB9300:DRAW"S8"
1070 RETURN
1100 CLS:FORX=0TO7:DRAW"BM"+STR
$(X*16+70)+",0;S8"+NU$(X+1):NEX
TX
1110 LINE(64,12)-(192,140),2,BF
1120 FORX=0TO8:LINE(X*16+64,12)
-(X*16+64,140),1:NEXT
1130 FORY=0TO8:LINE(64,Y*16+12)
-(192,Y*16+12),1:NEXT
1140 FORY=0TO7:DRAW"C15S8BM54,"
+STR$(Y*16+14)+NU$(Y+1):NEXT
1150 RETURN
```

```
1000 PS=0:CS=0
1010 FOR F=1 TO 8:FOR G=1 TO 8
1030 IF B(F,G)=1 THEN PAINT(G*1
6+54,F*16),1,3:PS=PS+1
1040 IF B(F,G)=2 THEN PAINT(G*1
6+54,F*16),4,3:CS=CS+1
1050 NEXT G,F
1060 DRAW"BM0,150;C2":A$=" VOCE
  ":GOSUB 9300:DRAW"BM186,150":
A$="COMPUT":GOSUB 9300
1063 LINE(62,150)-(70,158),PSET
,B:LINE(246,150)-(256,158),PSET
,B
1064 PAINT(248,152),4,2
1065 COLOR 1:LINE(0,60)-(32,80)
,PSET,BF:LINE(216,60)-(248,80),
PSET,BF
1066 DRAW"BM0,60;S16;C4":A$=MID
$(STR$(PS),2):GOSUB 9300:DRAW"B
M216,60":A$=MID$(STR$(CS),2):GO
SUB 9300:DRAW"S8"
1070 RETURN
1100 PCLS1:COLOR2:FORX=0 TO 7:D
RAW"BM"+STR$(X*16+70)+",0;S8"+N
U$(X+1):NEXT X
1110 LINE(64,12)-(192,140),PSET
,BF:COLOR 3
1120 FOR X=0 TO 8:LINE(X*16+64,
12)-(X*16+64,140),PSET:NEXT
1130 FOR Y=0 TO 8:LINE(64,Y*16+
12)-(192,Y*16+12),PSET:NEXT
1140 FOR Y=0 TO 7:DRAW"C2S8BM54
,"+STR$(Y*16+14)+NU$(Y+1):NEXT
1150 RETURN
```

A sub-rotina que tem início na linha 1000 desenha o tabuleiro. Primeiro, limpa a tela, imprime (ou desenha, no caso dos microcomputadores das linhas MSX e TRS-Color) os números das colunas e zera os dois placares.

O laço **FOR...NEXT** que fica entre as linhas 1010 e 1050 do programa desenha o tabuleiro na tela, linha por linha. À medida que as peças são mostradas, incrementa-se o placar.

As linhas 1052 e 1054 melhoram a "gramática" do placar, definindo se a palavra PONTO será usada no singular

ou no plural. A linha 1060, por sua vez, mostra o placar para o jogador.

No micro TRS-Color e no MSX, as linhas 1100 a 1150 constituem comandos extras de alta resolução para o desenho do tabuleiro.

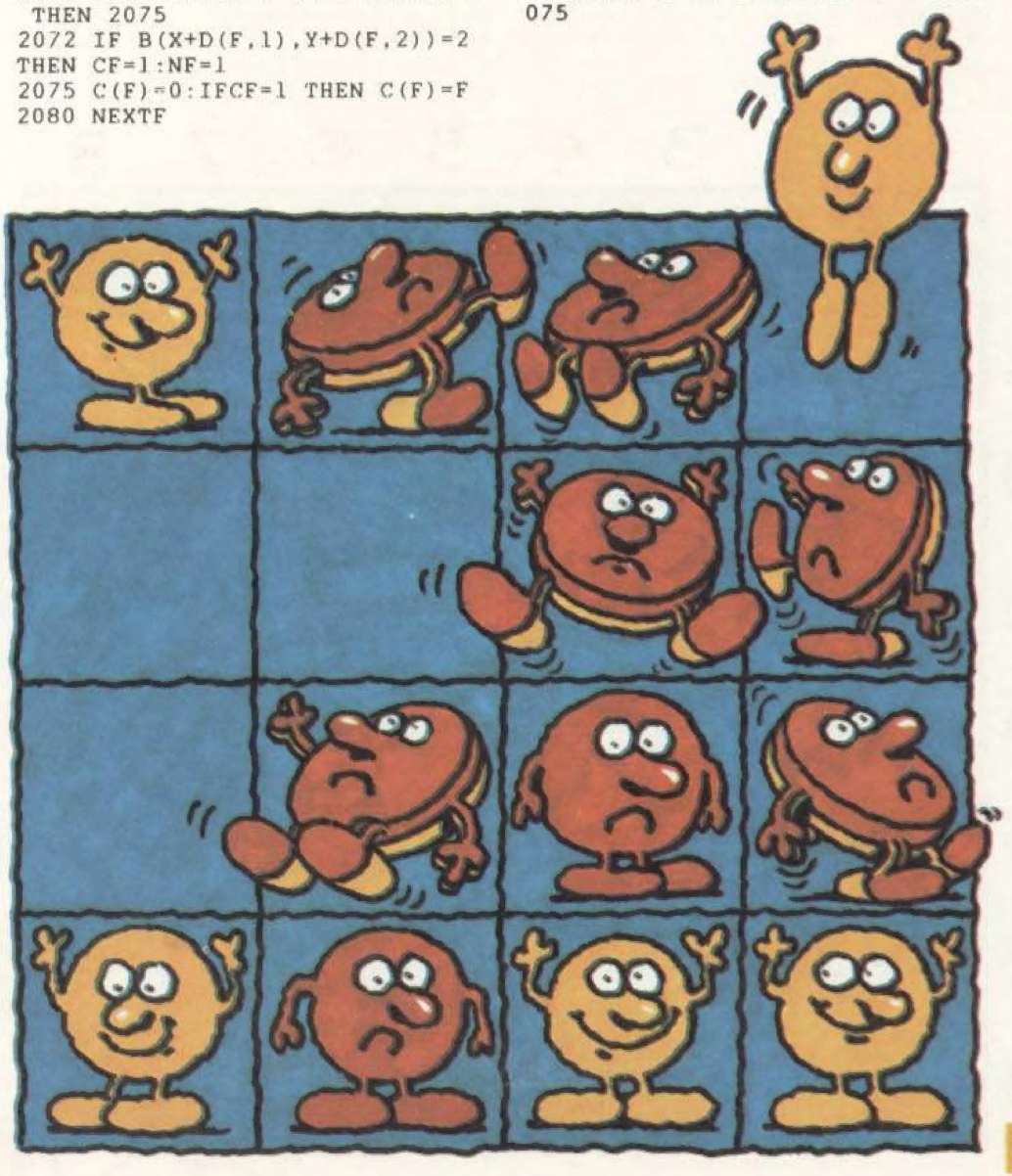
A VEZ DO JOGADOR

```
2000 PRINT AT 14,0;"FACA O MOVI
MENTO (LINHA,COLUNA)": INPUT X,
Y
2005 IF X=0 THEN  LET EG=1: RET
URN
2006 IF X=9 THEN  LET CP=2: RET
URN
2010 IF X<1 OR X>8 OR Y<1 OR Y>
8 THEN  GOTO 2000
2020 IF B(X,Y)=0 THEN  GOTO 207
0
2040 PRINT AT 17,0;" VOCE NAO P
ODE IR PARA UMA CASA  OCUPADA!"
: FOR F=1 TO 500: NEXT F
2050 PRINT AT 17,0;"
                                "
: GOTO 2000
2070 LET NF=0: FOR F=1 TO 8: LE
T CF=0: IF X+D(F,1)=0 OR X+D(F,
1)=9 THEN  GOTO 2075
2071 IF Y+D(F,2)=0 OR Y+D(F,2)=
9 THEN  GOTO 2075
2072 IF B(X+D(F,1),Y+D(F,2))=2
THEN  LET CF=1: LET NF=1
2075 LET C(F)=0: IF CF=1 THEN
LET C(F)=F
2080 NEXT F
2090 STOP
```

```
2000  VTAB (15): INPUT "QUAL
 SUA JOGADA (LINHA,COLUNA) ? ";
X,Y
2005  IF X = 0 THEN EG = 1: RET
URN
2006  IF X = 9 THEN CP = 2: RET
URN
2010  IF (X < 1 OR X > 8) OR (Y
 < 1 OR Y > 8) THEN 2000
2020  IF B(X,Y) = 0 THEN 2070
2040  VTAB (17): PRINT "ESTE LU
GAR JA' ESTA' OCUPADO": FOR F =
 0 TO 1500: NEXT
2050  VTAB (17): PRINT "
                    ": GOTO 20
00
2070 NF = 0: FOR F = 1 TO 8:CF
= 0: IF X + D(F,1) = 9 OR X + D
(F,1) = 0 THEN 2075
2071  IF Y + D(F,2) = 9 OR Y +
D(F,2) = 0 THEN 2075
2072  IF B(X + D(F,1),Y + D(F,2
)) = 2 THEN CF = 1:NF = 1
2075 C(F) = 0: IF CF = 1 THEN C
(F) = F
2080  NEXT F
2090  STOP
```

```
2000 LINE(0,182)-(255,191),1,BF
:LINE(112,150)-(144,180),1,BF:D
RAW"C15;BM10,182":A$="SUA JOGAD
A LINHA E COLUNA":GOSUB9300
2001 I$=INKEY$:IFI$<"0" OR I$>"
9"THEN2001
2002 X=VAL(I$):DRAW"BM114,154;S
16;C15"+NU$(X)
2003 I$=INKEY$:IFI$<"0" OR I$>"
9"THEN2003
2004 Y=VAL(I$):DRAWNU$(Y)+"S8"
2005 IFX=0 THEN EG=1:RETURN
2006 IFX=9 THEN CP=2:RETURN
2010 IFX<1 OR X>8 OR Y<1 OR Y>8
 THEN 2000
2020 IF B(X,Y)=0 THEN 2070
2040 LINE(0,182)-(255,191),1,BF
:DRAW"S8C15BM50,182":A$="QUADRA
DO OCUPADO":GOSUB9300:FORF=1TO9
00:NEXTF
2050 GOTO 2000
2070 NF=0:FORF=1TO8:CF=0:IFX+D(
F,1)=0 OR X+D(F,1)=9 THEN 2075
2071 IFY+D(F,2)=0 OR Y+D(F,2)=9
 THEN 2075
2072 IF B(X+D(F,1),Y+D(F,2))=2
THEN CF=1:NF=1
2075 C(F)=0:IFCF=1 THEN C(F)=F
2080 NEXTF
```

```
2000 COLOR 1:LINE(0,182)-(255,1
91),PSET,BF:LINE(118,150)-(150,
180),PSET,BF:DRAW"C3;BM0,182":A
$="PARA ONDE    LINHA E COLUNA"
:GOSUB 9300:SOUND 100,1
2001 I$=INKEY$:IF I$<"0" OR I$>
"9" THEN 2001
2002 X=VAL(I$):DRAW"BM118,150;S
16;C4"+NU$(X)
2003 I$=INKEY$:IF I$<"0" OR I$>
"9" THEN 2003
2004 Y=VAL(I$):DRAW NU$(Y)+"S8"
2005 IF X=0 THEN EG=1:RETURN
2006 IF X=9 THEN CP=2:RETURN
2010 IF X<1 OR X>8 OR Y<1 OR Y>
8 THEN 2000
2020 IF B(X,Y)=0 THEN 2070
2040 COLOR 1:LINE (0,182)-(255,
191),PSET,BF:DRAW"S8C4BM0,182":
A$="VOCE NAO PODE IR PARA AI":G
OSUB 9300:FOR F=1 TO 900:NEXT F
2050 GOTO 2000
2070 NF=0:FOR F=1 TO 8:CF=0:IF
X+D(F,1)=0 OR X+D(F,1)=9 THEN 2
075
```



```
2071 IF Y+D(F,2)=0 OR Y+D(F,2)=
9 THEN 2075
2072 IF B(X+D(F,1),Y+D(F,2))=2
THEN CF=1:NF=1
2075 C(F)=0:IF CF=1 THEN C(F)=F
2080 NEXT F
```

Sua jogada está entre as linhas 2000 e 2270, mas, por enquanto, o programa vai somente até a linha 2090 (2080, no MSX e no TRS-Color).

O programa obtém as coordenadas X e Y do jogador na linha 2000. A linha 2005 verifica se X é zero; se for, o indicador de fim de jogo é setado.

A linha 2010 verifica se houve um eventual erro de entrada das coordenadas do jogador; se houve, volta para a linha 2000. O programa checa, então, o quadrado escolhido, para se certificar de que está vazio, e salta para a linha 2070. Caso o quadrado já esteja ocupado por outra peça, a linha 2040 imprime uma mensagem de erro.

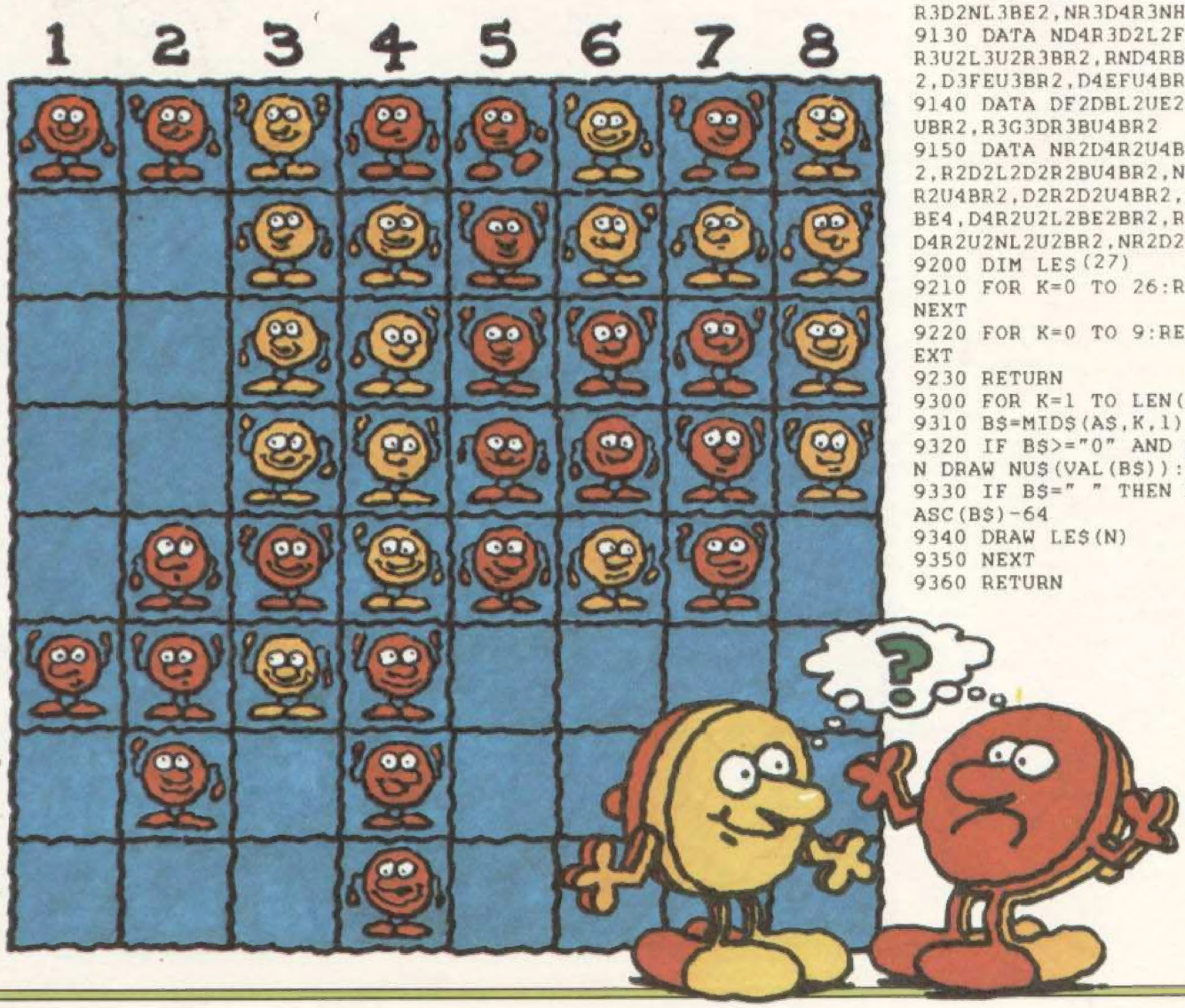
As linhas finais desta parte do programa (2070 a 2075) verificam se a peça em questão está ou não próxima de uma das peças do computador.

ROTINA PARA DESENHAR LETRAS

Para que possamos escrever na tela de alta resolução do MSX e do TRS-Color, precisaremos acrescentar esta rotina. Ela é bem semelhante à que foi descrita no artigo da página 232.

```
9100 DATA BR4,ND4R3D2NL3ND2BE2,
ND4R3DGNL2FDNL3BU4BR2,NR3D4R3BU
4BR2,ND4R2FD2GL2BE4BR,NR3D2NR2D
2R3BU4BR2
9110 DATA NR3D2NR2D2BE4BR,NR3D4
R3U2LBE2BR,D4BR3U2NL3U2BR2,ND4B
R2,BD4REU3L2R3BR2,D2ND2NF2E2BR2
9120 DATA D4R3BU4BR2,ND4FREND4B
R2,ND4F3DU4BR2,NR3D4R3U4BR2,ND4
R3D2NL3BE2,NR3D4R3NHU4BR2
9130 DATA ND4R3D2L2F2BU4BR2,BD4
R3U2L3U2R3BR2,RND4RBR2,D4R2U4BR
2,D3FEU3BR2,D4EFU4BR2
9140 DATA DF2DBL2UE2UBR2,DFND2E
UBR2,R3G3DR3BU4BR2
9150 DATA NR2D4R2U4BR2,BDEND4BR
2,R2D2L2D2R2BU4BR2,NR2BD2NR2BD2
R2U4BR2,D2R2D2U4BR2,NR2D2R2D2L2
BE4,D4R2U2L2BE2BR2,R2ND4BR2,NR2
D4R2U2NL2U2BR2,NR2D2R2D2U4BR2
9200 DIM LE$(26)
9210 FOR K=0 TO 26:READ LE$(K):
NEXT
9220 FOR K=0 TO 9:READ NU$(K):N
EXT
9230 RETURN
9300 FOR K=1 TO LEN(A$)
9310 B$=MID$(A$,K,1)
9320 IF B$>="0" AND B$<="9" THE
N DRAW NU$(VAL(B$)):GOTO 9350
9330 IF B$=" " THEN N=0 ELSE N=
ASC(B$)-64
9340 DRAW LE$(N)
9350 NEXT
9360 RETURN
```

```
9100 DATA BR4,ND4R3D2NL3ND2BE2,
ND4R3DGNL2FDNL3BU4BR2,NR3D4R3BU
4BR2,ND4R2FD2GL2BE4BR,NR3D2NR2D
2R3BU4BR2
9110 DATA NR3D2NR2D2BE4BR,NR3D4
R3U2LBE2BR,D4BR3U2NL3U2BR2,ND4B
R2,BD4REU3L2R3BR2,D2ND2NF2E2BR2
9120 DATA D4R3BU4BR2,ND4FREND4B
R2,ND4F3DU4BR2,NR3D4R3U4BR2,ND4
R3D2NL3BE2,NR3D4R3NHU4BR2
9130 DATA ND4R3D2L2F2BU4BR2,BD4
R3U2L3U2R3BR2,RND4RBR2,D4R2U4BR
2,D3FEU3BR2,D4EFU4BR2
9140 DATA DF2DBL2UE2UBR2,DFND2E
UBR2,R3G3DR3BU4BR2
9150 DATA NR2D4R2U4BR2,BDEND4BR
2,R2D2L2D2R2BU4BR2,NR2BD2NR2BD2
R2U4BR2,D2R2D2U4BR2,NR2D2R2D2L2
BE4,D4R2U2L2BE2BR2,R2ND4BR2,NR2
D4R2U2NL2U2BR2,NR2D2R2D2U4BR2
9200 DIM LE$(27)
9210 FOR K=0 TO 26:READ LE$(K):
NEXT
9220 FOR K=0 TO 9:READ NU$(K):N
EXT
9230 RETURN
9300 FOR K=1 TO LEN(A$)
9310 B$=MID$(A$,K,1)
9320 IF B$>="0" AND B$<="9" THE
N DRAW NU$(VAL(B$)):GOTO 9350
9330 IF B$=" " THEN N=0 ELSE N=
ASC(B$)-64
9340 DRAW LE$(N)
9350 NEXT
9360 RETURN
```



CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR LINHA

LINHA	FABRICANTE	MODELO
Apple II +	Appletronica	Thor 2010
Apple II +	CCE	MC-4000 Exato
Apple II +	CPA	Absolutus
Apple II +	CPA	Polaris
Apple II +	Digitus	DGT-AP
Apple II +	Dismac	D-8100
Apple II +	ENIAC	ENIAC II
Apple II +	Franklin	Franklin
Apple II +	Houston	Houston AP
Apple II +	Magnex	DM II
Apple II +	Maxitronica	MX-2001
Apple II +	Maxitronica	MX-48
Apple II +	Maxitronica	MX-64
Apple II +	Maxitronica	Maxitronic I
Apple II +	Microcraft	Craf II Plus
Apple II +	Milmar	Apple II Plus
Apple II +	Milmar	Apple Master
Apple II +	Milmar	Apple Senior
Apple II +	Omega	MC-400
Apple II +	Polymax	Maxxi
Apple II +	Polymax	Poly Plus
Apple II +	Spectrum	Microengenho I
Apple II +	Spectrum	Spectrum ed
Apple II +	Suporte	Venus II
Apple II +	Sycomig	SIC I
Apple II +	Unitron	AP II
Apple II +	Victor do Brasil	Elppa II Plus
Apple II +	Victor do Brasil	Elppa Jr.
Apple IIe	Microcraft	Craft IIe
Apple IIe	Microdigital	TK-3000 IIe
Apple IIe	Spectrum	Microengenho II
MSX	Gradiente	Expert GPC-1
MSX	Sharp	Hotbit HB-8000
Sinclair Spectrum	Microdigital	TK-90X
Sinclair Spectrum	Timex	Timex 2000
Sinclair ZX-81	Apply	Apply 300
Sinclair ZX-81	Engebras	AS-1000
Sinclair ZX-81	Filcres	NEZ-8000
Sinclair ZX-81	Microdigital	TK-82C
Sinclair ZX-81	Microdigital	TK-83
Sinclair ZX-81	Microdigital	TK-85
Sinclair ZX-81	Prologica	CP-200
Sinclair ZX-81	Ritas	Ringo R-470
Sinclair ZX-81	Timex	Timex 1000
Sinclair ZX-81	Timex	Timex 1500
TRS-80 Mod. I	Dismac	D-8000
TRS-80 Mod. I	Dismac	D-8001/2
TRS-80 Mod. I	LNW	LNW-80
TRS-80 Mod. I	Video Genie	Video Genie I
TRS-80 Mod.III	Digitus	DGT-100
TRS-80 Mod.III	Digitus	DGT-1000
TRS-80 Mod.III	Kemitron	Naja 800
TRS-80 Mod.III	Prologica	CP-300
TRS-80 Mod.III	Prologica	CP-500
TRS-80 Mod.III	Sysdata	Sysdata III
TRS-80 Mod.III	Sysdata	Sysdata Jr.
TRS-80 Mod.IV	Multix	MX-Compacto
TRS-80 Mod.IV	Sysdata	Sysdata IV
TRS-Color	Codimex	CS-6508
TRS-Color	Dynacom	MX-1600
TRS-Color	LZ	Color 64
TRS-Color	Microdigital	TKS-800
TRS-Color	Prologica	CP-400

CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR NOME

FABRICANTE	MODELO	PAÍS	LINHA
Appletronica	Thor 2010	Brasil	Apple II +
Apply	Apply 300	Brasil	Sinclair ZX-81
CCE	MC-4000 Exato	Brasil	Apple II +
CPA	Absolutus	Brasil	Apple II +
CPA	Polaris	Brasil	Apple II +
Codimex	CS-6508	Brasil	TRS-Color
Digitus	DGT-100	Brasil	TRS-80 Mod.III
Digitus	DGT-1000	Brasil	TRS-80 Mod.III
Digitus	DGT-AP	Brasil	Apple II +
Dismac	D-8000	Brasil	TRS-80 Mod. I
Dismac	D-8001/2	Brasil	TRS-80 Mod. I
Dismac	D-8100	Brasil	Apple II +
Dynacom	MX-1600	Brasil	TRS-Color
ENIAC	ENIAC II	Brasil	Apple II +
Engebras	AS-1000	Brasil	Sinclair ZX-81
Filcres	NEZ-8000	Brasil	Sinclair ZX-81
Franklin	Franklin	USA	Apple II +
Gradiente	Expert GPC1	Brasil	MSX
Houston	Houston AP	Brasil	Apple II +
Kemitron	Naja 800	Brasil	TRS-80 Mod.III
LNW	LNW-80	USA	TRS-80 Mod. I
LZ	Color 64	Brasil	TRS-Color
Magnex	DM II	Brasil	Apple II +
Maxitronica	MX-2001	Brasil	Apple II +
Maxitronica	MX-48	Brasil	Apple II +
Maxitronica	MX-64	Brasil	Apple II +
Maxitronica	Maxitronic I	Brasil	Apple II +
Microcraft	Craft II Plus	Brasil	Apple II +
Microcraft	Caft IIe	Brasil	Apple IIe
Microdigital	TK-3000 IIe	Brasil	Apple IIe
Microdigital	TK-82C	Brasil	Sinclair ZX-81
Microdigital	TK-83	Brasil	Sinclair ZX-81
Microdigital	TK-85	Brasil	Sinclair ZX-81
Microdigital	TK-90X	Brasil	Sinclair Spectrum
Microdigital	TKS-800	Brasil	TRS-Color
Milmar	Apple II Plus	Brasil	Apple II +
Milmar	Apple Master	Brasil	Apple II +
Milmar	Apple Senior	Brasil	Apple II +
Multix	MX-Compacto	Brasil	TRS-80 Mod.IV
Omega	MC-400	Brasil	Apple II +
Polymax	Maxxi	Brasil	Apple II +
Polymax	Poly Plus	Brasil	Apple II +
Prologica	CP-200	Brasil	Sinclair ZX-81
Prologica	CP-300	Brasil	TRS-80 Mod.III
Prologica	CP-400	Brasil	TRS-Color
Prologica	CP-500	Brasil	TRS-80 Mod.III
Ritas	Ringo R-470	Brasil	Sinclair ZX-81
Sharp	Hotbit HB-8000	Brasil	MSX
Spectrum	Microengenho I	Brasil	Apple II +
Spectrum	Microengenho II	Brasil	Apple IIe
Spectrum	Spectrum ed	Brasil	Apple II +
Suporte	Venus II	Brasil	Apple II +
Sycomig	SIC I	Brasil	Apple II +
Sysdata	Sysdata III	Brasil	TRS-80 Mod.III
Sysdata	Sysdata IV	Brasil	TRS-80 Mod.IV
Sysdata	Sysdata Jr.	Brasil	TRS-80 Mod.III
Timex	Timex 1000	USA	Sinclair ZX-81
Timex	Timex 1500	USA	Sinclair ZX-81
Timex	Timex 2000	USA	Sinclair Spectrum
Unitron	AP II	Brasil	Apple II +
Victor do Brasil	Elppa II Plus	Brasil	Apple II +
Victor do Brasil	Elppa Jr.	Brasil	Apple II +
Video Genie	Video Genie I	USA	TRS-80 Mod. I

UM LOGOTIPO PARA CADA MODELO DE COMPUTADOR

INPUT foi especialmente projetado para microcomputadores compatíveis com as sete principais linhas existentes no mercado.
Os blocos de textos e listagens de programas aplicados apenas a determinadas linhas de micros podem ser identificados por meio dos seguintes símbolos:

Sinclair ZX-81

TRS-80

TK-2000

MSX

Spectrum

TRS-Color

Apple II

Quando o emblema for seguido de uma faixa, então tanto o texto como os programas que se seguem passam a ser específicos para a linha indicada.

NO PRÓXIMO NÚMERO

PROGRAMAÇÃO BASIC
Probabilidade, freqüência e acaso. Programa lançador de moedas.
O triângulo de Pascal. Distribuição de freqüência.

CÓDIGO DE MÁQUINA
Ajude Willie em sua incansável busca do lanche roubado.
Aprenda ainda a chamar as sub-rotinas da memória ROM.

PROGRAMAÇÃO BASIC
Trajetória de projéteis. Efeitos de diferentes campos
gravitacionais. Simulação de parábolas.